UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Detección de Escenas de Violencia con Modelos Deep Learning

Ernesto Luis Bisbé
Tutor: Manuel Antonio Sánchez-Montañés Isla

Junio 2020

DETECCIÓN DE ESCENAS DE VIOLENCIA CON MODELOS DEEP LEARNING

AUTOR: Ernesto Luis Bisbé
TUTOR: Manuel Antonio Sánchez-Montañés Isla

Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2020

Resumen (castellano)

Este Trabajo de Fin de Grado propone desarrollar un sistema de detección automática de escenas violentas. En primer lugar, se realiza un estudio del estado del arte examinando las distintas herramientas técnicas disponibles para el análisis de vídeo, concentrando este trabajo en el empleo de técnicas de aprendizaje automático, en concreto el aprendizaje profundo (*deep learning*). Se compararán distintos modelos convolucionales profundos con el objetivo de entender las ventajas y desventajas de estas técnicas de análisis y su aplicación en el caso de reconocimiento de escenas de violencia. Se usan modelos tanto totalmente desarrollados como modelos basados en transferencia de aprendizaje (*transfer learning*) con el objetivo de mejorar la calidad de la red entrenada. Se procede a perfeccionar estos modelos con técnicas que se apoyan en otros campos de aprendizaje profundo para mejorar su capacidad, y por último se somete a examen el modelo en juegos de datos (*Datasets*) públicos como: *MoviesFight* y *HockeyFight*, con el objetivo de medir su tasa de acierto y entendimiento cualitativo del modelo. Por último, se revisan futuras perspectivas de investigación que surgen a partir de las conclusiones de este trabajo.

Abstract (english)

In this Bachelor Thesis it is proposed to develop an automatic violent scenes detector. On a first approximation, a state-of-the-art is carried out, examining the different technical tools available for video analysis, focusing on this thesis on machine learning techniques, and specifically on deep learning. It compares different deep learning models in order to understand the advantages and disadvantages of this analysis techniques and their application in the detection of violent scenes. It's uses both already developed models and others based on transfer learning to improve the quality of the trained network. Proceeds to improve those models with procedures constructed on other areas of deep learning to improve their learning capabilities, and eventually test these models on public Datasets made for this problem, such as MoviesFight and HockeyFight, with the aim to measure the accuracy and qualitative capabilities of the model. Finally, it reviewed future research topics that branch from the findings of this thesis.

Palabras clave (castellano)

Reconocimiento de imagen, violencia, red neuronal, convolución, red residual, *ResNet*, *LSTM*, *HockeyFight*, *MoviesFight*, *ViolentFlows*, *RWF-2000, Keras, TensorFlow*, mapas de calor

Keywords (inglés)

Image recognition, violence, neural network, convolution, residual network, ResNet, LSTM, HockeyFight, MoviesFight, ViolentFlows, RWF-2000, Keras, TensorFlow, heatmaps

Agradecimientos

Gracias a mi padre, por su apoyo y la increíble inspiración que proporciona la dedicación plena tanto al trabajo como al bienestar de todo el que aparece por su vida.

Gracias a mi madre, por su cariño y pasión en cada uno de mis proyectos, por el sacrificio hecho durante décadas que nos ha permitido convertirnos en la mejor versión posible de nosotros mismos.

Gracias a mi hermano, por ser el mejor amigo que puedo tener, por tener siempre un oído disponible al que hablar, tener un sitio donde reír, tener un sitio donde llorar.

Gracias a Patri, por ser la persona maravillosa que eres, por aportar tanto en nuestro tan corto horizonte al demostrar la titánica fuerza de voluntad y buen humor que te caracteriza.

Gracias a mis amigos, por aguantar mis delirios de programación de madrugada y poder tener otro hogar, vaya donde vaya.

Gracias a Manuel, por guiarme durante este proyecto, tan desconocido al principio, y hacer la carga menos pesada.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

1.1 Motivación

La idea principal que motiva esta investigación es conseguir avanzar en el diseño, desarrollo e integración de un sistema automático, teniendo en cuenta los avances tecnológicos relacionados con la inteligencia artificial de los últimos años. Dicho sistema ha de ser capaz de detectar-segmentar escenas de vídeo cuyo contenido sea violento, en una amplia variedad de situaciones, ya sea en acontecimientos deportivos, como contenido cinematográfico, o violencia callejera en cámaras de seguridad. Para ello se parte de un sistema básico existente y representativo del estado del arte utilizado para análisis de vídeo. La tarea consiste en mejorar la efectividad del diseño original, intentando solventar las carencias observadas en los resultados del modelo inicial. Las soluciones optimizadas, resultantes del presente trabajo, están inspiradas en técnicas usadas en otros problemas de reconocimiento de patrones en imagen. Todo ello debidamente integrado, formará parte de este Trabajo de Fin de Grado (TFG).

1.2 Objetivos

El objetivo principal de este proyecto consiste en entrenar un modelo básico de aprendizaje profundo, implementando conocimiento a partir de técnicas de mejora de aprendizaje, y de esta forma, conseguir un modelo final, capaz de servir de soporte a las tareas de detección automática de escenas de vídeo en las que se recoja violencia. Para conseguir este fin se ha de:

1. Estudiar la información sobre modelos y técnicas existentes para poder establecer la base correcta del desarrollo del proyecto.
2. Establecer premisas que permitan evaluar el funcionamiento adecuado de los modelos y garanticen que el análisis de los resultados sea veraz, en términos estadísticos.
3. Comparar los métodos estudiados para poder establecer con cuál de ellos se obtienen mejores resultados.
4. Vislumbrar limitaciones en el modelo desarrollado y platear posibles líneas de continuidad en la investigación.

Para la elaboración del documento de memoria se ha presupuesto que la información general existente acerca de estas técnicas y de esta aplicación en particular no es abundante, por lo que se explicará con el mayor grado de detalle posible tanto el procedimiento general como la conceptualización previa, de forma que se facilite su comprensión a todo interesado.
Los puntos por discutir en este proyecto se organizan de la siguiente manera:

i Estado del arte para el problema de reconocimiento de imágenes.
ii Diseño e implementación del modelo base.
iii Datos aumentados y relación con el aprendizaje.
iv Análisis de las funciones de aprendizaje inductivo regularizado (*kernel regularizers*).
v Estudio del empleo de las funciones de marginado (*dropout*) y su relación con el aprendizaje.
vi Resultados del modelo final y análisis de resultados.

1.3 Organización de la memoria

La memoria consta de los siguientes contenidos:

Capítulo 1: Introducción: motivación, objetivos y organización.

Capítulo 2: Estado del arte: antecedentes, breve descripción de las características y propiedades de las redes y modelos que han servido de base en el proyecto, sobre el análisis de imagen y vídeo. Revisión histórica. Conceptos generales. El aprendizaje automático y aprendizaje profundo, como base para el desarrollo de este sistema. Criterios de selección empleados.

Capítulo 3: Desarrollo del proyecto: descripción detallada del modelo base, su diseño y arquitectura en cada una de las etapas. Debilidades del modelo base. Introducción de mejoras al modelo con el objetivo de subsanar dichas debilidades. Módulos de entrenamiento y Análisis. Integración. Juegos de datos utilizados. Elementos para establecer valoraciones.

Capítulo 4: Experimentación, elaboración de pruebas mediante conjuntos de datos seleccionados. Comparativa de resultados obtenidos por el sistema mejorado.

Capítulo 5: Conclusiones obtenidas tras el análisis de resultados y recomendaciones para dar continuidad al trabajo.

Referencias: Listado los artículos científicos revisados, debidamente referenciados en el cuerpo del trabajo. Licencias de uso de los juegos de datos.

Glosario: Siglas, terminología en inglés y breve descripción de las funciones desarrolladas.

Anexos: Código del modelo implementado compartimentado en funciones de entrenamiento y análisis.

2 Estado del arte

2.1 Introducción

En el campo de la clasificación de acciones en imágenes se distinguen dos etapas separadas: la extracción de características y la clasificación [1]**.** Mientras que la primera define la información que describe inequívocamente las particularidades de cada imagen, la segunda consiste en una separación de elementos en distintos grupos en función de sus características con el objetivo de separar estos grupos de la manera más eficiente posible.

Los primeros métodos implementados para la detección de escenas se basaban en el establecimiento de características predefinidas con la ayuda de algoritmos de regresión [2]. Con los años estos métodos fueron evolucionado hasta el momento, en que con las redes neuronales aparece el concepto de inteligencia artificial, mediante la aplicación de métodos de detección de acción, basados en el autoaprendizaje de características [3] (Figura 2-1).

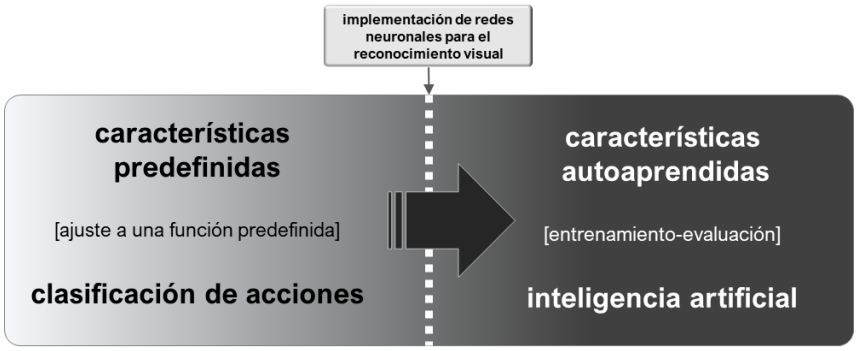

Figura 2-1: Proceso evolutivo del reconocimiento de imágenes

En este apartado se tratará el estado del arte en el problema de detección de escenas violentas. Se introducirán los rasgos que poseen las escenas violencias, para después hacer una revisión sobre las técnicas de detección de acciones.

2.2 Detección de acción en vídeo con características predefinidas

Históricamente, las primeras aproximaciones al análisis de imagen y vídeo se basaban en realizar un análisis exhaustivo de las características locales de la imagen mediante la utilización de histogramas de gradientes orientados (*HOG, Histogram of Oriented Gradients*) [4], histogramas de flujos ópticos orientados (*HOOF, Histogram of Oriented Optic Flow*) [5], patrones binarios locales (*LBP, Local Binary Patterns*) [6], histogramas de perímetro de movimiento (*MBH¸ Motion Boundary Histogram*) [7] o uso de la ondícula de Haar [2] como nuevo origen de espacio de características.

Estos algoritmos extraen características concretas de la imagen, y mediante un clasificador se podrán usar para poder detectar elementos de acción. Se comentarán distintos clasificadores en el apartado 2.4.2. Para el problema de clasificación de acción uno de los clasificadores más usados fue la máquina de vectores de soporte (*SVM, Support Vector Machines*) [8].

Algunos de los trabajos más destacados en esta aproximación son los siguientes:

- Giannakopoulos y colaboradores [9] intentaron una aproximación multimodal con un clasificador de vecinos más próximos (*k-NN, k-nearest neighbors*) utilizando características basadas en texto y audio en la imagen, pero sacrificando reconocimiento sofisticado;

- Nievas y colaboradores [10] utilizaron técnicas de transformación de características invariantes a escala adaptadas a movimiento (*MoSIFT*) [11] en conjunto con un clasificador *SVM*, que proporcionarán los mejores resultados en esta área debido a la complejidad del desarrollo;
- Eyben y colaboradores [12] diseñaron un clasificador segmental a gran escala junto a otra *SVM*, basándose en clasificaciones audiovisuales y análisis de vídeo a bajo nivel con el uso de histogramas, pero no consiguió resultados concluyentes;
- *Mironică* y colaboradores [13] plantearon un sistema basado en extraer un vector de descriptos totalmente agregados y clasificarlos con un *SVM*, haciendo una extracción de características a nivel de fotograma (*frame)*, lo que implicaba una gran carga computacional.

El principal problema del uso de características predefinidas para la clasificación es que no se puede alterar la característica elegida según las clases y las imágenes, por lo que si se ha escogido una característica que no es adecuada para distinguir las categorías en un problema en concreto, la efectividad del clasificador baja significativamente. Por lo tanto, ha sido común la práctica de utilizar múltiples características e intercalarlas para conseguir un mejor rendimiento, lo cual requiere de una labor manual ardua, además de una falta de previsión de múltiples heurísticas, lo que le resta propósito al intento de generar un clasificador universal [14].

En la siguiente sección se expondrán los sistemas basados en construir características mediante *Machine Learning*, que tratan de solventar este problema extrayendo de manera automática las características óptimas en función del problema a resolver.

2.3 Aprendizaje automático

2.3.1 Revisión histórica

La ambición de crear un sistema que simule el cerebro humano alimentó el desarrollo inicial de las redes neuronales. En 1943, McCulloch y Pitts intentaron comprender cómo el cerebro podía producir patrones altamente complejos mediante el uso de células básicas interconectadas, llamadas neuronas. El modelo McCulloch y Pitts de una neurona, llamado modelo ***MCP***, significó uno de los principales aportes al desarrollo de redes neuronales artificiales (ver epígrafe 2.4.2). En la Tabla 1 se presenta una serie de contribuciones importantes en este campo [15], que incluyen por ejemplo *LeNet* y *LSTM* (*Long Short Term Memory)*, que conducen a la "era del aprendizaje profundo" de hoy.

Otro hito importante fue el desarrollo del algoritmo de entrenamiento basado en la propagación hacia atrás de errores o retropropagación (***backpropagation***) [16,17] es un método de cálculo del gradiente utilizado en algoritmos de aprendizaje supervisado utilizados para entrenar redes neuronales artificiales. El método emplea un ciclo propagación – adaptación de dos fases. Una vez que se ha aplicado un patrón a la entrada de la red como estímulo, este se propaga desde la primera capa a través de las capas siguientes de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas. El prototipo ***LeNet*** nace en 1989 [18] a partir un requerimiento de la identificación de códigos postales escritos a mano, por parte del servicio de correos de los Estados Unidos. *LeNet* fue una de las primeras redes neuronales convolucionales y promovió el desarrollo del aprendizaje profundo. Como casi todas las demás redes neuronales, están entrenados con una versión del algoritmo de retropropagación. Donde difieren es en la arquitectura.

Tabla 1. Algunos hitos importantes en la historia de las redes neuronales.

	Hito	Descripción básica	Desarrollador, año
Neuronas	***MCP***	neurona considerada como el antepasado de la red neuronal artificial.	McCulloch & Pitts, 1943
	Hebbian learning rule	regla de aprendizaje que especifica cuánto se debe aumentar o disminuir el peso de la conexión entre dos unidades en proporción al producto de su activación.	Hebb, 1949
	Perceptron	primera neurona artificial o unidad básica de inferencia de discriminador lineal.	Rosenblatt, 1958
	Backpropagation	método de cálculo del gradiente utilizado en algoritmos de aprendizaje supervisado utilizados para entrenar redes neuronales artificiales.	Werbos, 1974 Rumelhart, Hinton, y Williams(1986) [16]
	Neocognitron	red neuronal artificial jerárquica de varias capas, considerado como el antepasado de la red neuronal convolucional.	Fukushima, 1980 [36]
	Boltzmann Machine	tipo de red neuronal recurrente estocástica.	Ackley, Hinton y Sejnowski, 1985
	Restricted Boltzmann Machine (Harmonium)	red neuronal artificial estocástica generativa que puede aprender una distribución de probabilidad sobre su conjunto de entradas.	Smolensky, 1986
	Recurrent Neural Network	red neuronal que integra bucles de realimentación, permitiendo a través de ellos que la información persista durante algunos pasos o épocas de entrenamiento, a través conexiones desde las salidas de las capas, que "incrustan" sus resultados en los datos de entrada.	Jordan, 1986
	Autoencoders	tipo de red neuronal artificial que se utiliza para aprender codificaciones de datos eficientes de manera no supervisada.	Rumelhart, Hinton y Williams, 1986 yBallard, 1987
RNC	***LeNet***	estructura de red neuronal convolucional.	LeCun, 1989 [18]
Aprendizaje profundo	***LSTM***	es una arquitectura artificial de red neuronal recurrente utilizada en el campo del aprendizaje profundo.	Hochreiter y Schmidhuber, 1997 [22]
	Deep Belief Network	modelo gráfico generativo, o alternativamente un tipo de red neuronal profunda, compuesta por múltiples capas de variables latentes, con conexiones entre las capas, pero no entre unidades dentro de cada capa.	Hinton, 2006 [30]
	Deep Boltzmann Machine	modelo gráfico probabilístico no dirigido con múltiples capas de variables aleatorias ocultas.	Salakhutdinov y Hinton, 2009 [27]
	AlexNet	red neuronal convolucional.	Krizhevsky, Sutskever, y Hinton, 2012 [35]

Las redes neuronales convolucionales están diseñadas para reconocer patrones visuales directamente de imágenes de píxeles con un preprocesamiento mínimo. Pueden reconocer patrones con una variabilidad extrema (como caracteres escritos a mano) y con solidez a las distorsiones y transformaciones geométricas simples.

En 1998, LeCun y colaboradores [19] hacen una revisión de varios métodos aplicados al reconocimiento de caracteres escritos a mano y los compararon con las referencias existentes. Los resultados mostraron que las redes neuronales convolucionales (*LeNet-5*) superaban a todos los demás modelos. *LeNet-5* como se muestra en la Figura 2-2 consta de siete capas (datos de entrada: imagen 32×32 píxeles) [20,21].

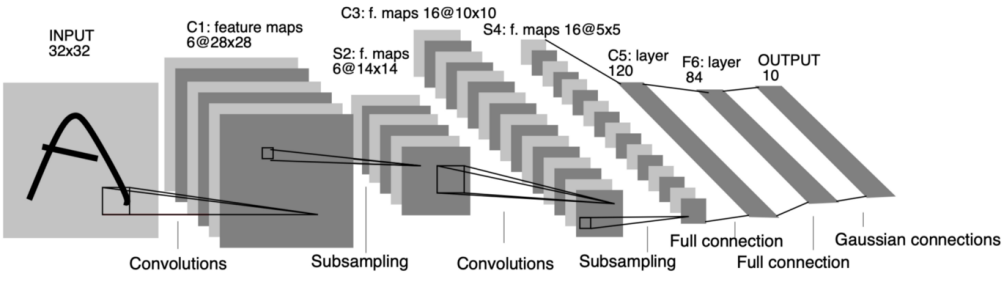

Figura 2-2: Arquitectura de *LeNet-5* [20]

No hace falta decir que la cobertura de este trabajo a nivel histórico, no es exhaustiva; por ejemplo, las redes *Long Short Term Memory* (***LSTM*)** [22] en la categoría de redes neuronales recurrentes, son eficientes en cuanto a la predicción y detección de patrones en secuencias, lo que las hace muy atractivas en este tipo de contexto. Por otro lado, las *LSTM* son un tipo de redes recurrentes. Como se ha explicado previamente, la característica principal de las redes recurrentes es que la información puede persistir introduciendo bucles en su diagrama por lo que, básicamente, pueden "recordar" estados previos y utilizar esta información para decidir cuál será el siguiente. Esta característica las hace muy adecuadas para manejar series temporales. Mientras las redes recurrentes estándar pueden modelar dependencias a corto plazo (es decir, relaciones cercanas en la serie temporal), las *LSTM* están diseñadas con el fin de aprender dependencias largas, por lo que se podría decir que tienen una "memoria" a más largo plazo [23]. Estas redes se utilizan a la hora de diseñar modelos de redes neuronales para cumplir la función de clasificación. Se aplican predominantemente en problemas como el modelado del lenguaje, clasificación de texto, reconocimiento de escritura a mano, traducción automática, reconocimiento de voz/ música [24, 25, 26].

Uno de los avances más importantes en el aprendizaje profundo se produjo en 2006, cuando Hinton y colaboradores introdujeron las ***DBM, Deep Boltzmann Machine*** [27], con múltiples capas de *RBM, Restricted Boltzmann Machine*, entrenando con avidez una capa a la vez de una manera no supervisada. Guiar la capacitación de los niveles intermedios de representación utilizando el aprendizaje no supervisado, realizado localmente en cada nivel, fue el principio principal detrás de una serie de desarrollos que provocaron el aumento de la última década en arquitecturas profundas y algoritmos de aprendizaje profundo.

Entre los factores más destacados que contribuyeron al gran impulso del aprendizaje profundo se encuentran la aparición de grandes conjuntos de datos etiquetados disponibles públicamente, de gran calidad, junto con las unidades de procesamiento de gráficos (*GPU, Graphics Processing Units)*. Todo ello permitió la transición de las implementaciones basadas en unidades de procesamiento central (*CPU, Central Processing Units)* a la *GPU*, permitiendo así para una aceleración significativa en el entrenamiento de modelos profundos.

Hay otros factores que también han desempeñado papeles importantes en el desarrollo de *Deep Learning* como la solución del problema del *vanishing gradient* (usando funciones de activación como *ReLU* en vez de funciones de tipo sigmoidal como la tangente hiperbólica o la función logística, la propuesta de nuevas técnicas de regularización [28] (por ejemplo, *dropout, batch normalization*, y *data augmentation*), y la aparición de marcos de trabajo (*frameworks*) potentes, que permiten una creación de prototipos de forma más rápida. El aprendizaje profundo ha impulsado grandes avances en una variedad de problemas de visión por ordenador, como la detección de objetos, el seguimiento del movimiento, el reconocimiento de acciones, estimación de pose humana y segmentación semántica [15].

En este capítulo se explicarán los principales desarrollos en arquitecturas de aprendizaje profundo y algoritmos para aplicaciones de visión por ordenador. En este contexto, se explicarán brevemente tres de los tipos más importantes de modelos de aprendizaje profundo con respecto a su aplicabilidad en la comprensión visual:

- Redes de creencias profundas "familia Boltzmann"
- ***Autoencoders denoising***
- Redes neuronales convolucionales (*CNN*)

Las ***RBM, Restricted Boltzmann Machine*** [29] son redes estocásticas generativas que aprenden la distribución de los datos de entrada. Se llaman restringidas porque no permiten

conexiones entre nodos de su capa oculta. Las redes ***DBN, Deep Belief Networks*** están formadas de varias *RBM + finetuning* con *backpropagation* y *gradient descent.* Por su parte las ***DBM, Deep Boltzmann Machine****s* también están formadas de varias *RBM + finetuning* con *backpropagation* y *gradient descent.* Ambas son modelos de aprendizaje profundo que pertenecen a la "familia Boltzmann", ya que utilizan la ***RBM*** como módulo de aprendizaje [30]. Como se aprecia en la Figura 2-3, la diferencia entre ambas redes *DBM* y *DBN* es que la *DBN* es dirigida y la *DBM* no lo es [31].

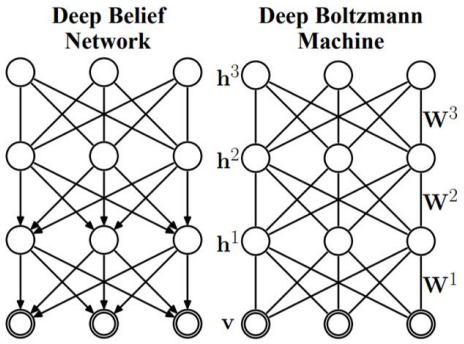

Figura 2-3: Diferencia entre DBN y DBM (3 capas) [31]

Los ***autoencoders*** [32] son redes neuronales que se usan comúnmente para la selección y extracción de funciones. Sin embargo, cuando hay más nodos en la capa oculta que entradas, la red se puede aprender la llamada *Identity Function*, también llamada *Null Function*, lo que implica que la salida es igual a la entrada, y provoca que el *Autoencoder* sea inútil. Los ***autoencoders denoising*** resuelven este problema corrompiendo los datos de entrada a propósito, dándole valor 0 de forma aleatoria a algunos de los valores. En general, el porcentaje de nodos de entrada que se establecen en cero es aproximadamente del 50%. Otras fuentes sugieren un recuento más bajo, como 30%, dependiendo de la cantidad de datos y nodos de entrada que tenga [33].

Las **Redes Neuronales Convolucionales** (*CNN, Convolutional Neural Networks*) se inspiraron en la estructura del sistema visual. Los primeros modelos computacionales estaban basados en estas conectividades locales entre neuronas y en transformaciones jerárquicamente organizadas de la imagen. Son capaces de detectar características simples como por ejemplo detención de bordes, líneas, siluetas y componer en características más complejas hasta detectar lo que se busca (detalle sobre convolución en epígrafe 2.3.4).

La arquitectura que ha representado con mayor claridad la potencia de las redes neuronales convolucionales fue ***AlexNet.*** *AlexNet* [34, 35] es el nombre de una red neuronal convolucional (*CNN*), diseñada por Alex Krizhevsky, publicada junto con Ilya Sutskever y Geoffrey Hinton. El 30 de septiembre de 2012, *AlexNet* compitió en el desafío de reconocimiento visual a gran escala sobre la *ImageNet* (base de datos visual diseñada para la investigación de *software* de reconocimiento visual). La red logró un error del 15,3%, más de 10,8 puntos porcentuales por debajo de su sucesor, ganando el concurso. El resultado principal de la investigación original fue que la profundidad del modelo era esencial para conseguir alto rendimiento. Aun cuando computacionalmente era costoso, resultaba factible debido a la utilización de unidades de procesamiento de gráficos durante el entrenamiento. *AlexNet* no fue la primera implementación rápida en *GPU* de una *CNN* en ganar un concurso de reconocimiento de imágenes (Figura 2-4). Una *CNN* en *GPU* por Chellapilla y colaboradores [36] fue 4 veces más rápida que una implementación equivalente en la CPU. Una *CNN* profunda de Ciresan y colaboradores [37] en IDSIA ya era 60 veces más rápida y logró un rendimiento sobrehumano en agosto de 2011. Entre mayo de 2011 y septiembre de 2012, su *CNN* ganó al menos de cuatro concursos de

imágenes. También mejoraron significativamente el mejor rendimiento en la literatura para múltiples bases de datos de imágenes.

AlexNet como implementación es “similar" la red de Ciresan, ambas fueron escritas originalmente con *CUDA* para poder ejecutarse con soporte de *GPU*. De hecho, en realidad ambas son sólo variantes de los diseños de *CNN* introducidos por Y. LeCun y colaboradores [18] que aplicaron el algoritmo de retropropagación a una variante de la arquitectura *CNN* original de K. Fukushima llamada *neocognitron* [38]. La arquitectura fue modificada posteriormente por el método de J. Weng llamado *MaxPooling* [39].

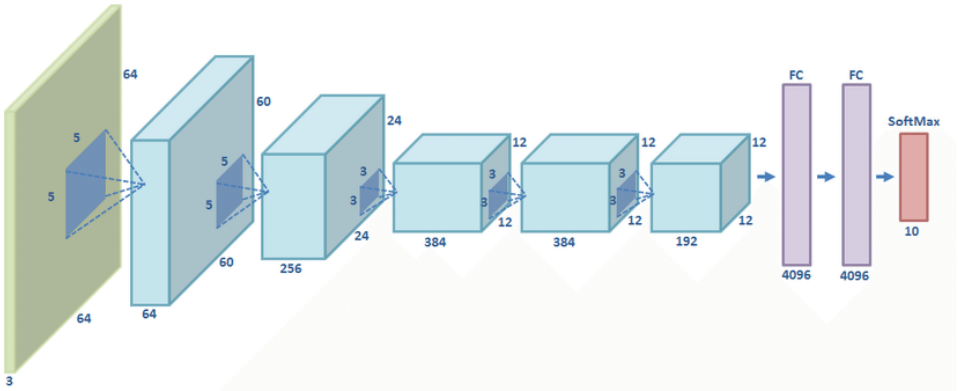

Figura 2-4: Esquema de la red *AlexNet* [35]

La implementación de *AlexNet* contenía ocho capas: las primeras cinco eran capas convolucionales, algunas de ellas seguidas por capas de agrupamiento máximo, y las últimas tres eran capas completamente conectadas. La publicación donde se introducía *AlexNet* es considerada una de las publicaciones más influyentes en visión artificial, ya que ha estimulado la aceleración del aprendizaje profundo. En 2015 *AlexNet* fue superada por una *CNN* profunda de Microsoft (>100 capas) [40], que ganó el concurso *ImageNet2015*.

2.3.2 Conceptos básicos de aprendizaje automático

Se define aprendizaje automático [41] como el campo de la inteligencia artificial en el que se diseñan sistemas que modifican su respuesta en función de conocimiento previo, de manera análoga a la que un humano aprende, utilizando como base la experiencia.

El aprendizaje automático *machine learning* [42] implementa además algoritmos de optimización matemática que, mediante aproximaciones sucesivas, hacen converger una solución. Este proceso de optimización ocurre mediante funciones de pérdida que, de manera cuantitativa, definen cuán errónea puede ser la predicción del modelo, respecto al resultado verdadero. Se definen varios tipos de algoritmos de aprendizaje automático, en función de la aproximación al problema, los tipos de entrada y salida de los datos y del problema que quieren resolver. La separación clásica de algoritmos se basa en conocer: si el aprendizaje del algoritmo es supervisado, es decir, una vez este realiza la predicción, se le indica si ha acertado o fallado en esta, con el objetivo de que pueda autoevaluarse o si, por el contrario, el algoritmo es no supervisado, significa que no se conoce el resultado final de la predicción. Además, hay más tipos básicos, no sólo esos dos: aprendizaje semisupervisado, y aprendizaje por refuerzo [43]. Las aplicaciones del primero se encuentran en obtención de modelos matemáticos lineales o no lineales de manera empírica (a partir de los propios datos del problema) y el segundo está más relacionado con la detección de estructuras en el *Dataset*, sin tener como referencia una variable objetivo.

Tanto en los algoritmos supervisados como en los no supervisados, el proceso por el que se realizan aproximaciones sucesivas al problema mediante la ingesta de datos y la comprobación del error de predicción se le llama “fase de entrenamiento”. Su objetivo es ir

regulando los parámetros del algoritmo de tal forma que la función de pérdida vaya reduciendo (minimizando) y, como consecuencia, la tasa de acierto de la predicción vaya aumentando (maximizando). Una vez concluida la fase de entrenamiento (*training)*, se pasa a la fase de validación.

Los procesos de trabajo se pueden dividir en tres tipos:

- **Entrenamiento**: adiestrará al algoritmo.
- **Validación:** proceso mediante el cual se evalúa un modelo con un conjunto diferente de datos. La validación se lleva a cabo tras el entrenamiento y junto con este, persigue el objetivo de optimizar el modelo, en términos de rendimiento.
- **Prueba o *test***, aislado de la fase de entrenamiento, comprobará el rendimiento del modelo en datos nuevos para él.

La creación de estos conjuntos permite detectar ciertos problemas típicos a la hora de entrenar los modelos:

Sobreajuste *(overfitting)*: El modelo está aprendiendo características exclusivas del conjunto de entrenamiento que no tienen carácter general, por lo que, a la hora de comprobar los resultados con el conjunto de validación o prueba se observará un rendimiento peor de lo esperado.

Infraajuste *(underfitting)*: Ocurre cuando el modelo sea demasiado sencillo para el problema a resolver o quizá que el modelo aprenda muy lentamente, por temor a sobreajuste, lo que ocasiona que no termine de converger el algoritmo.

En general, uno de los problemas que tienen estos algoritmos de aprendizaje supervisado es que es necesario construir las características que se quieren analizar del conjunto de datos (*feature engineering*). Sin embargo, se han desarrollado técnicas que permiten al algoritmo buscar de manera autónoma a qué características debe dar importancia para maximizar la función de pérdida.

Algunos de los algoritmos supervisados de *Machine Learning* más importantes son:

Árboles de decisión: modelo de predicción utilizado en diversos ámbitos que van desde la inteligencia artificial hasta la economía. Dado un conjunto de datos se fabrican diagramas de construcciones lógicas, muy similares a los sistemas de predicción basados en reglas, que sirven para representar y categorizar una serie de condiciones que ocurren de forma sucesiva, para la resolución de un problema [44].

Regresión logística: modelo de regresión utilizado para predecir el resultado de una variable categórica en función de las variables independientes o predictoras. Es útil para modelar la probabilidad de un evento ocurriendo como función de otros factores. Están basados en propiedades estadísticas de los datos [45].

Máquinas de soporte vectorial (*SVM, Support Vector Machines):* métodos propiamente relacionados con problemas de clasificación y regresión. Intuitivamente, una *SVM* es un modelo que representa a los puntos de muestra en el espacio, separando las clases a 2 espacios lo más amplios posibles mediante un hiperplano de separación definido como el vector entre los 2 puntos de las 2 clases, más cercanos al que se llama vector soporte. Cuando las nuevas muestras se ponen en correspondencia con dicho modelo, en función de los espacios a los que pertenezcan, pueden ser clasificadas a una u otra clase, por tanto, es un modelo basado en propiedades geométricas de los datos [46].

Redes Bayesianas o red de creencia: modelo bayesiano (de Bayes) o modelo probabilístico en un grafo acíclico dirigido, es un modelo grafo probabilístico que representa un conjunto de variables aleatorias y sus dependencias condicionales a través de un grafo acíclico dirigido (*DAG, Directed Acyclic Graph*) [47]

Redes neuronales artificiales: modelo matemático desarrollado en este proyecto, por lo

tanto, resulta imprescindible describir sus componentes principales y su principio de funcionamiento, comenzando por el concepto de neurona artificial.

En principio, se define una **neurona artificial** [48] como un programa que recibe diversas entradas, ya sean externas o de otras neuronas, las combina y produce una salida. Mientras que la definición clásica de neurona se entiende como una suma de contribuciones lineales regulada por una función de activación a la salida (Figura 2-5), hay diferentes tipos de neuronas que modifican su comportamiento para conseguir una mejor adaptación a problemas específicos.

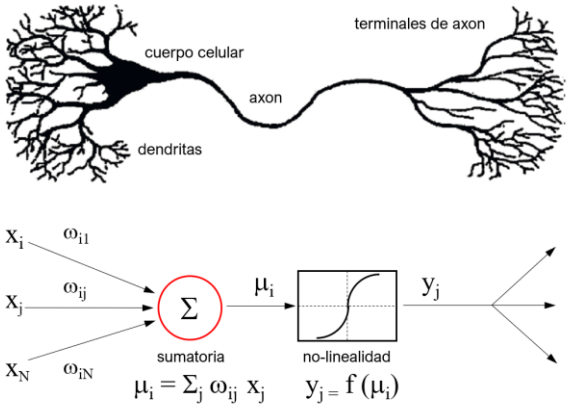

Figura 2-5: Neurona artificial

Dentro de estas adaptaciones posibles se encuentran: cambiar la operación de combinación de valores de entrada (usar un operador convolucional en vez de un sumatorio. Se detallará más adelante en el apartado 2.4.3) o normalizar y reescalar los valores de entrada, esto último denominado "normalización por lotes" (*BN, Batch Normalization*) [49].

Después de que las entradas sean alteradas por los pesos entrenados de la neurona, los valores resultantes serán, la entrada de una función, que regularizará el resultado de la neurona con el fin de normalizar el valor de salida. A esta función se le denomina función de activación. La función de activación combina el potencial postsináptico que proporciona la función de propagación con el estado actual de la neurona para conseguir el estado futuro de activación de dicha neurona. Normalmente, esta función es monótona creciente, pudiéndose citar las más comunes:

- **Sigmoide** [50]: muchos procesos naturales y curvas de aprendizaje de sistemas complejos muestran una progresión temporal desde unos niveles bajos al inicio, hasta acercarse a un clímax transcurrido un cierto tiempo. La transición se produce en una región caracterizada por una fuerte aceleración intermedia. La función sigmoide permite describir esta evolución. Su gráfica tiene una típica forma de "S". En general, la función sigmoide se refiere al caso particular de la función logística y que viene definida por la expresión:

  $P_{(t)} = 1/(1+e^{-t})$ (1)

  donde:
  $P_{(t)}$: denotada como población
  e: constante de Euler
  t: tiempo

- ***SoftMax* o función exponencial normalizada** [51]: es una generalización de la función logística. Se emplea para "comprimir" un vector *k*-dimensional, *z*, de valores reales arbitrarios en un vector *k*-dimensional, $\sigma(z)$, de valores reales en el rango [0, 1]. La función está dada por:

$z \rightarrow \sigma(z) : R^k \rightarrow [0, 1]^k$ (2)

donde:
z: vector k-dimensional
σ : función *SoftMax*
R: números reales
k: dimensión

- **Unidad lineal rectificadora** (*ReLU, Rectifier Linear Unit*) [52]: es una función de activación, también es conocida como "función rampa" y es análoga a la rectificación de media onda en electrónica. Es definida como:

  $f(x) = max (0, x)$ (3)

  donde:
  x: es la entrada de la neurona

Una vez comprendido el funcionamiento de una neurona, se puede definir entonces una **red neuronal artificial** (*ANN, Artificial Neural Networks*) [35] o sistemas conexionistas, basados en las redes biológicas neuronales que constituyen la mente animal, estableciendo nodos y sinapsis de información de un nodo a otro (Figura 2-6). Cada nodo circular representa una neurona artificial y cada flecha representa una conexión desde la salida de una neurona a la entrada de otra.

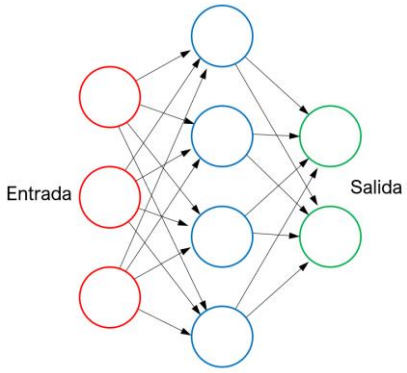

Figura 2-6: Red neuronal artificial

Una red neuronal artificial es, básicamente, un algoritmo automático estructurado o jerárquico que emula el aprendizaje humano con el fin de obtener ciertos conocimientos. Destacan porque no requieren de reglas programadas previamente, sino que el propio sistema es capaz de "aprender" por sí mismo para efectuar una tarea a través de una fase previa de entrenamiento: aprendizaje de características o aprendizaje de representación (*feature learning*). En redes estándares, una neurona recibe uno o varios elementos de entrada, y su salida es una función lineal o no lineal de la suma de sus entradas. En estas funciones que realiza cada neurona, la contribución de las entradas se define mediante un peso. Son estos pesos los valores que se van regulando durante el aprendizaje del modelo. Estas neuronas se organizan en capas y una neurona puede o no estar conectada con cualquier otra de otra capa. A mayor cantidad de capas, mayor es la profundidad de la red y, presuntamente mayor es la capacidad de aprendizaje [53].

Las redes neuronales artificiales están conformadas por tres tipos de capas:

- **Capa de entrada:** reciben la información desde el exterior de la red (*input*).
- **Capa de salida:** envían la información hacia el exterior de la red (*output*).
- **Capa o capas ocultas (opcionales):** transmiten la información entre los nodos de la red *(hidden)*. Por lo tanto, se encuentran en el medio de los nodos de entrada y de salida y no tienen contacto con el exterior.

En este contexto, los nodos de entrada reciben una serie de datos desde el exterior, que son enviados al interior de la red. La información se va procesando y transfiriendo de una capa a otra. Este proceso es lo que se conoce como "aprendizaje" pues cada capa de nodos va aprendiendo en cada iteración**.** Cuando las redes neuronales son entrenadas, cada red crea, modifica o elimina conexiones entre los nodos con el fin de dar respuestas más acertadas ante el problema que busca resolver. En función de la arquitectura de conexión, de unos nodos con otros y del sentido de la información, existen diferentes tipos de redes neuronales.

El objetivo del diseño en red es separar el análisis de información en partes, de tal manera que cada elemento se vuelva experto en un apartado del problema en concreto, creando de esta forma un sistema autónomo crítico con el problema, que pueda comprender variaciones a dicho problema.

Las neuronas tienen una información de entrada (*kernel*) y un sesgo (*bias)* que es independiente de la misma, permitiéndole regular de manera autónoma la importancia de la información que porta la neurona, sin necesidad de afectar a los pesos de esta.

Las redes neuronales se pueden diferenciar a partir de distintos elementos:

- Las neuronas o nodos, que constituyen el elemento básico de procesamiento.
- La arquitectura de la red descrita por las conexiones ponderadas entre los nodos.
- El algoritmo de entrenamiento, usado para encontrar los parámetros de la red.

Una vez definida la arquitectura que se desea utilizar en un problema particular, la red neuronal debe ajustarse a una muestra dada, a través del proceso de aprendizaje.

Las redes neuronales de alimentación positiva (*Feed-forward ANN*) [54] se corresponden a la clase de *ANN* más estudiada por el ámbito científico y la más utilizada en los diversos campos de aplicación. Su característica principal es que cada capa alimenta exclusivamente a la capa siguiente, no permitiendo dependencias de una capa consigo misma, ni con capas anteriores. Existen redes neuronales que poseen conexiones en bucle, llamadas "redes neuronales recurrentes" (*RNN, Recurrent Neural Networks*) [55]. Esto permite su empleo en aplicaciones de carácter temporal, ya que poseen una salida dinámica que posibilita al modelo de adquisición de memoria. Un proyecto que utilice redes neuronales también puede complementarse con otros algoritmos tradicionales. Existen numerosas soluciones de compromiso entre los algoritmos de aprendizaje. Casi cualquier algoritmo va a funcionar bien, con los parámetros correctos para la formación de un conjunto específico de datos fijos. Sin embargo, la selección y el ajuste de un algoritmo para la formación en datos no previstos requieren una cantidad significativa de experimentación. Si se seleccionan apropiadamente el modelo, la función de coste y el algoritmo de aprendizaje, la red neuronal artificial resultante puede ser extremadamente robusta (Figura 2-7).

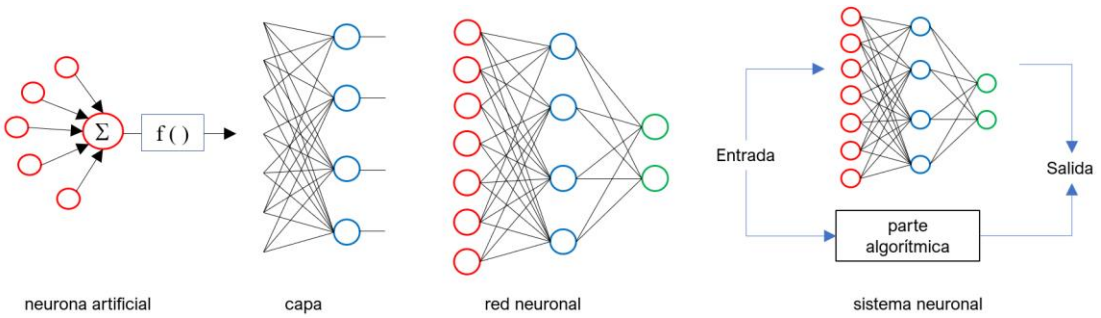

Figura 2-7: Diferentes niveles de procesamiento en una red neuronal artificial [56]

2.3.3 Aprendizaje profundo

El aprendizaje profundo (*deep learning*) es un subcampo dentro de las redes neuronales artificiales. Este entra en funcionamiento, cuando dichas redes poseen varias ocultas. En aproximaciones iniciales se demostró que un perceptrón lineal, no es apto como clasificador universal, mientras que una red con una función de activación no polinómica con más de una capa oculta, puede serlo [56]. Como regla general, al contar con altos niveles de profundidad, el aprendizaje profundo le permite al sistema aprender elementos cada vez más complejos. De esta manera, ayuda a que los ordenadores no solamente aprendan significados sino también a que comprendan simbolismos, contextos completos y dinámicos.

¿Cómo funciona el aprendizaje profundo?

Por ejemplo, se quiere que una máquina sea capaz de identificar si hay algún perro dentro de una imagen [57]. Para ello sería necesario programar un algoritmo de una manera semejante al esquema de la Figura 2-8.

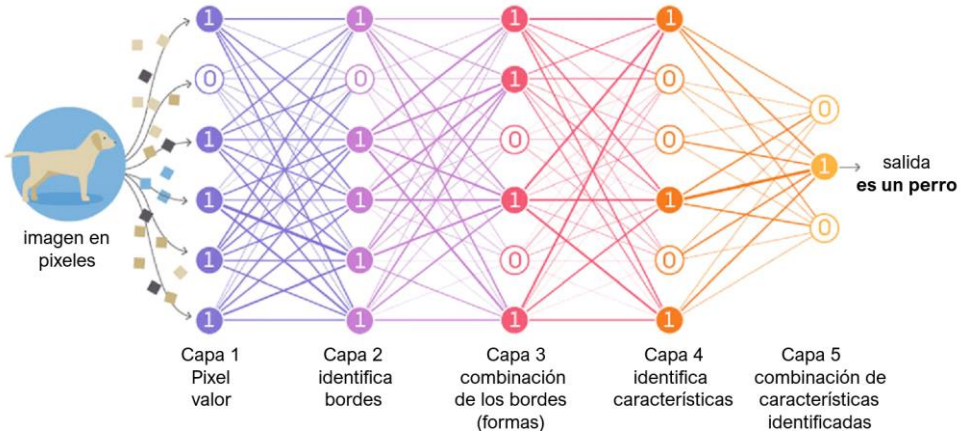

Figura 2-8: Ejemplo de aprendizaje profundo [57]

Para la entrada de datos se tendría que crear una capa que asimile la información introducida. Después, la capa de segundo nivel tiene como objetivo procesar cada uno de los píxeles delimitando los bordes dentro de estos (separando los vectores dentro de los píxeles). En el tercer nivel se combinarían los bordes para diseñar las formas y constituir cada uno de los objetos de la imagen. En la capa del cuarto nivel, se utilizan los filtros del sistema para reconocer qué objetos son perros, y cuáles no, identificando, por ejemplo, cuando aparezcan cuatro patas, una cola y un hocico. Como último paso, la capa 4 traspasa los datos a la última capa, la cual combina las características identificadas para reconocerse si es un perro o no por medio de conclusiones parciales. Es decir, este fragmento es una cola de un animal, por tanto, sí puede ser un perro; si tiene cuatro patas, sí tiene características de perro, etc. así hasta entregar todos los fragmentos de información a la capa de salida y que este ofrezca una conclusión. En general, puede decirse que el aprendizaje profundo funciona reduciendo errores y tratando de aumentar el intervalo de confianza. Si fuese necesario basar la conclusión utilizando solamente la segunda capa, se puede decir que el intervalo de confianza de que haya un perro es de 70%, luego, si lo procesa la tercera capa aumentaría hasta el 77%, con una cuata capa se sobrepasaría el 85%… así hasta reducir el margen de error casi a 0. Cabe destacar que para que la máquina aprenda tiene que pasar por un proceso de aprendizaje el cual combina un aprendizaje supervisado (un humano etiqueta en la imagen que es un perro), y un aprendizaje no supervisado (la máquina encuentra sus propios patrones para establecer relaciones a partir de los datos aportados). Mientras más cerca esté la neurona de la capa de salida, más

entrenamiento supervisado requerirá para perfeccionarse. Es preciso realizar un trabajo de diseño de arquitectura para estas redes neuronales profundas que estará en función de la problemática que se quiera abordar, donde cada diseño se especializará en resolver un problema en concreto.

2.3.4 Redes neuronales convolucionales

La convolución [58] es una operación matemática en la que se calcula la integral del producto de dos funciones donde una de ellas es trasladada e invertida en la variable o variables sobre las que se integra. El resultado es función de la traslación aplicada. Debido a esto, tiene la propiedad de mostrar cómo la forma de una función es modificada por otra. La convolución no se aplica solo a funciones del tiempo, sino también funciones del espacio (1D, 2D, 3D). Un ejemplo de aplicación de un *kernel* sobre un conjunto de datos observa en la Figura 2-9. [84]

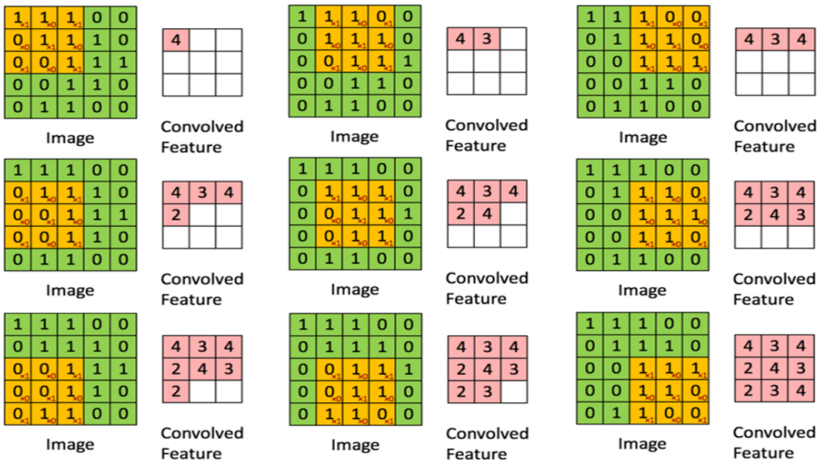

Figura 2-9: Ejemplo de operación de convolución [84]

Una red neuronal convolucional *CNN* comprende tres tipos principales de capas neurales, (i) capas convolucionales, (ii) capas de agrupación y (iii) capas completamente conectadas. Cada tipo de capa juega un papel diferente. Cada capa de una *CNN* transforma el volumen de entrada en un volumen de salida de activación neuronal, lo que finalmente conduce a las capas finales completamente conectadas, lo que resulta en un mapeo de los datos de entrada a un vector de características. Las *CNN* han sido extremadamente exitosas en aplicaciones de visión por ordenador, tales como: reconocimiento de rostros, detección de objetos, visión poderosa en robótica y autos sin conductor.

(i) Capas convolucionales (*convolutional layers*). En las capas convolucionales, una *CNN* utiliza varios *kernels* para convolucionar la imagen completa, así como los mapas de características intermedias, generando varios mapas de características. Debido a las ventajas de la operación de convolución, se ha propuesto como un sustituto de capas completamente conectadas con el fin de lograr tiempos de aprendizaje más rápidos.

(ii) Capas de agrupación (*pooling layers*). Las capas de agrupación se encargan de reducir las dimensiones espaciales (ancho × altura) del volumen de entrada para la siguiente capa convolucional. La capa de agrupación no afecta la dimensión de profundidad del volumen. La operación realizada por esta capa también se denomina submuestreo o disminución de muestreo, ya que la reducción de tamaño conduce a una pérdida simultánea de información. Sin embargo, dicha pérdida es beneficiosa para la red porque la disminución de tamaño conduce a una menor sobrecarga computacional para las próximas capas de la red, y también funciona contra el sobreajuste. La agrupación promedio y la agrupación máxima son las estrategias más utilizadas. El subtipo más

popular de capa se denomina *MaxPooling*, cuya salida es simplemente el valor máximo de todos los datos de entrada.

(iii) Capas totalmente conectadas (*FCL, fully-connected layers*). Después de varias capas convolucionales y de agrupación, el razonamiento de alto nivel en la red neuronal se realiza a través de capas completamente conectadas. Las neuronas en una capa totalmente conectada tienen conexiones completas a toda la activación en la capa anterior, como su nombre lo indica. Por lo tanto, su activación se puede calcular con una multiplicación de matriz seguida de un desplazamiento de sesgo. Esta estructura permite reducir el número total de salidas de la capa anterior, y se suelen usar en las etapas finales de la red. A este tipo de capas también se les denomina "densamente conectadas".

Las propiedades de convolución, trasladadas al dominio de la imagen, se pueden usar para el análisis de contornos y poder describir el contenido de las imágenes. En una red convolucional, este proceso ocurre sobre una serie de múltiples capas, y en cada una de ellas se realiza una convolución, sobre el resultado de la capa anterior. El modelo objeto de este estudio, se especializará en el uso de redes convolucionales. Las redes convolucionales [59] reales raramente se construyen solamente sobre capas convolucionales, sino que se combinan con capas como *MaxPooling* y *FCL* a la salida.

En las primeras etapas de investigación acerca del uso de redes convolucionales como herramientas de análisis de imagen, se estableció que un aumento de la profundidad de la red provocaba un aumento en la tasa de aprendizaje del modelo [60]. El problema que conlleva el aumento de esta profundidad es que en consecuencia el análisis que se hace en las primeras capas de la imagen va a ser cada vez más generalista debido a que el análisis específico del problema se realizará en las capas más profundas cuando se haga una comparación entre una imagen real y su predicción. Por lo tanto, los pesos de las capas iniciales apenas aprenderán. Este problema se denomina "gradiente desvaneciente" (*vanishing gradient)* y se encuentra debido a que las redes neuronales se entrenan mediante el algoritmo de retropropagación de errores (*backpropagation*), que depende del algoritmo de descenso por gradiente. Este algoritmo compara la salida deseada con la respuesta de la red y corrige en sentido contrario al de la red los pesos de las neuronas, con el fin de ir corrigiendo su error. Es habitual tratar de deducir los algoritmos de aprendizaje a partir de un cierto criterio a optimizar. Es decir, se debe de proponer un criterio que mida el rendimiento de la red neuronal para encontrar una regla de actualización de pesos que la optimice. Una manera extendida de definir el rendimiento es a partir del error cuadrático medio de las salidas actuales de la red respecto de las deseadas. La función que se computa para definir el valor del error total suele ser en el caso de problemas de clasificación la entropía cruzada (*categorical cross entropy*) [61].

El problema de *vanishing gradient* se soluciona, aportando como puntos de entrada del modelo parte de la información de manera directa, sin filtrar, lo que permite que las características encontradas en el modelo se aprendieran con respecto a una entrada establecida y que posibilita su utilización para otras aplicaciones. A las redes neuronales que aplican este tipo de soluciones se les denomina "redes residuales" debido a que transportan como residuo información sin procesar. Una arquitectura de red residual implica conexiones de atajos de identidad (*identity shortcut connections)*. Esto provoca que, en las primeras iteraciones del entrenamiento, se va a comprimir la información en pocas capas, debido a que se está proporcionando la información de manera directa. Sin embargo, a medida que se continúa iterando, las capas se irán expandiendo y las partes "residuales" de las capas irán explorando cada vez más del conjunto de características de la imagen original.

$$y = H(x, w) = f(x, w) + x \quad (4)$$

donde:
y: salida
x: entrada
w: pesos de la neurona

En estos casos, el gradiente se puede propagar hacia atrás usando directamente el atajo, lo cual permite que no se desvanezca. La implementación de *f(x, w)* en el caso de *ResNetV2* (Figura 2-10), es una aplicación de *Batch Normalization* a la entrada, una activación *ReLU* y una operación pudiendo ser convolucional en 2D (*Conv2D)* o *FCL*, aplicado 2 veces seguidas, con un sumatorio final. Puede sorprender que la activación se encuentre antes de la operación, se ha demostrado que resulta adecuado anticipar las activaciones a la operación, para evitar distorsiones en el espacio transformado (siempre no-negativas). Intuitivamente una función residual, puede tener valores tanto positivos como negativos [62] debido a que mantiene su estructura de entrada. Así que, *f(x, w)* se controlará con las funciones de activación en la zona intermedia, por lo que la salida no estará controlada por una función de activación. En la implementación original de *ResNet* (Figura 2-10) [40] se tiene una implementación clásica (operación, *BN*, *ReLU*, operación, *BN*, sumatorio, *ReLU*), pero por los motivos antes explicados, se ha probado que no es la arquitectura óptima para solucionar el problema de *vanishing gradients.* Luego se utiliza la modificación *ResNetV2* [60] que presupone la preactivación del peso, en vez de la la post-activación de la implementación clásica.

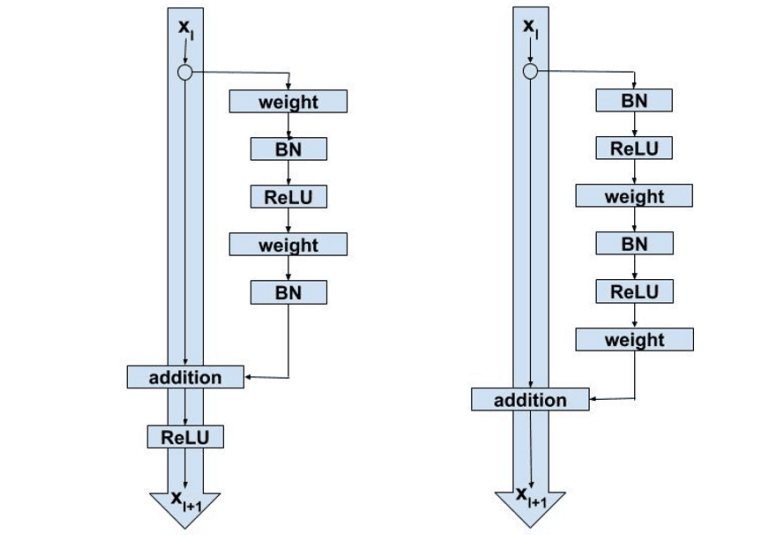

Figura 2-10: Implementación ResNet original y ResNetV2 [60]

2.3.5 Aplicaciones de *Deep Learning* en visión artificial

A continuación, se mencionan algunos trabajos que han aprovechado los métodos de aprendizaje profundo para abordar tareas clave en la visión por ordenador, como la clasificación de imágenes, la detección de objetos, el reconocimiento de rostros, el reconocimiento de acción y actividad, y la estimación de la postura humana [15].

Clasificación de imágenes: Es un proceso que consiste en otorgar una categoría / etiqueta a una imagen a partir del contenido de esta. Como este concepto es unos de los pilares principales de este trabajo, se detallará con mayor profundidad en el capítulo 2.3.6.

Detección de objetos: Es el proceso de detectar instancias de objetos semánticos de una clase determinada (como humanos, aviones o pájaros) en imágenes digitales y vídeo

(Figura 2-11). Un enfoque común para los marcos de detección de objetos incluye la creación de un gran conjunto de ventanas candidatas, que se clasifican en la secuela utilizando las características de *CNN*. Una gran cantidad de trabajos se basa en el concepto de regiones características [63]. Con este fin, tales métodos siguen una detección de objetos a partir de un enfoque de segmentación semántica, que generalmente logra buenos resultados.

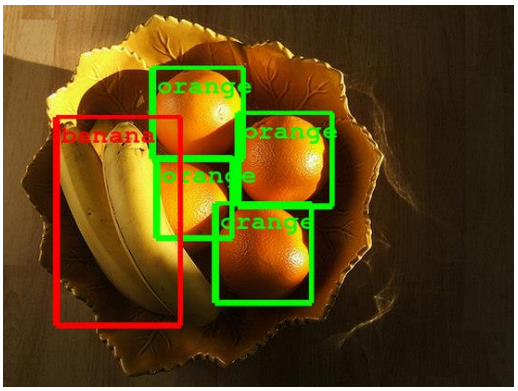

Figura 2-11: Detección de objetos en imágenes digitales en Google [63]

Reconocimiento facial. Es una aplicación de visión por ordenador con gran éxito comercial. Antes del desarrollo del *Deep Learning* se propusieron una variedad de sistemas de reconocimiento facial basados en la extracción de características que se definían de modo "artesanal". En tales casos, un extractor de características obtiene propiedades de una cara alineada para obtener una representación de baja dimensión, en base a la cual un clasificador hace predicciones [64]. Las *CNN* provocaron un cambio en el campo del reconocimiento facial, gracias a sus características de aprendizaje e invariancia de transformación. El primer trabajo que empleó *CNN* para el reconocimiento facial fue desarrollado por Lawrence y colaboradores [65]. Hoy día, los *CNN* ligeros y *VGG Face Descriptor* se encuentran entre los más modernos. *FaceNet* de Google [66] y *Deep-Face* de Facebook [67] se basan en *CNN*. *DeepFace* modela una cara en 3D y la alinea para que aparezca como una cara frontal.

Reconocimiento de acción y actividad. La acción humana y el reconocimiento de la actividad es un tema de investigación que ha recibido mucha atención de los investigadores. Por ejemplo, en [68] se muestra un sistema de aprendizaje profundo para la detección y el reconocimiento de eventos complejos en secuencias de vídeo: primero, se utilizaron mapas de prominencia (*salience*) para detectar y localizar eventos, y luego se aplicó el aprendizaje profundo a las características preentrenadas para identificar los cuadros más importantes que corresponden al evento subyacente.

Implementaciones exitosas de sistemas de detección de acciones son por ejemplo [69] donde se reconoce acciones en vídeos de voleibol de playa, y [70] en los que se reconoce actividad, a partir del uso de teléfonos inteligentes.

Estimación de pose humana. El objetivo de la estimación de la postura humana es determinar la posición de las articulaciones humanas a partir de imágenes, secuencias de imágenes, imágenes de profundidad o datos de esqueleto proporcionados por el hardware de captura de movimiento [71]. La estimación de la postura humana es una tarea compleja debido a la amplia variedad de formas y apariencias en las que puede aparecer una persona, además de la gran variedad de iluminaciones y fondos que puede haber. Antes de la era del aprendizaje profundo, la estimación de la postura se basaba en la detección de partes del cuerpo, por ejemplo, a través de estructuras pictóricas [72].

2.3.6 Aplicación de redes convolucionales para clasificación de imágenes

Tal y como se ha explicado, en casos de imágenes las redes convolucionales contienen varias capas ocultas, donde se pueden detectar: líneas y curvas. Así se van especializando hasta poder reconocer formas complejas como un rostro, bordes, siluetas, etc.

En el caso de tratamiento de imagen [73], mientras que una red de capas totalmente conectadas con alimentación positiva se puede usar para aprender características de una imagen aparte de clasificar información. No resulta práctico aplicar esta arquitectura, ya que requeriría un inmenso número de neuronas, incluso en una red superficial habría una gran cantidad de información asociada con las imágenes, donde cada píxel tendría información variable. Para una imagen de pequeño tamaño, una sola neurona poseería un número considerable de pesos. Sin embargo, el mismo análisis que utiliza el resultado de una convolución es posible con un número mucho menor de parámetros. Otra ventaja de aplicaciones de imágenes de las redes convolucionales es el bajo nivel de reprocesamiento necesario para evaluar las imágenes, en contra de otros algoritmos de clasificación de imágenes.

Un detalle importante en la aplicación de estas redes es la necesidad de que las neuronas de una capa estén localmente conectadas a la siguiente -en vez de totalmente conectadas- debido a que la correlación espacial que existe en las imágenes provoca que la información en espacios locales sea muy redundante. Por lo tanto, no es tan interesante un examen global de la imagen como sucesivos análisis locales. Una vez analizado las distintas aplicaciones que presentan las redes neuronales, en el campo de clasificación de acción se han investigado 4 métodos distintos, que se han aplicado con eficacia:

Clasificación con conjuntos de fotogramas (*frames*)

Karpathy y colaboradores [74] utilizan un modelo de red convolucional para analizar una secuencia de imágenes, probando tanto un fotograma como conjuntos de fotogramas, tanto conjuntos correlacionados temporalmente, como totalmente independientes (principio y final del vídeo). Los resultados muestran buen rendimiento en el caso de conjuntos de fotogramas correlacionados temporalmente (Figura 2-12).

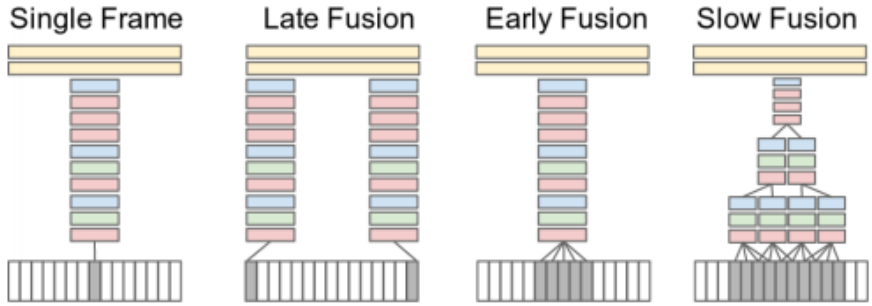

Figura 2-12: Clasificación con fotogramas (*frames*) [74]

Utilización de dos *CNN* en paralelo para reconocimiento de acción

Karen y colaboradores [75] emplean en paralelo dos *ConvNets* para el análisis de vídeo: una red convolucional que opera en fotogramas de manera individual, tratándolas como si fueran imágenes en solitario, junto a otra red convolucional que analiza el flujo óptico del vídeo [76,77], obteniendo el resultado final como una contribución de los resultados de las dos redes (Figura 2-13). Mientras que es una solución técnicamente efectiva, requiere una gran carga computacional debido a la necesidad de analizar cada fotograma de manera individual, por lo que no resulta ser una solución práctica.

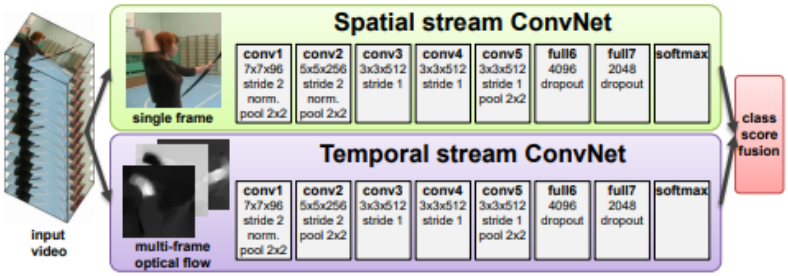

Figura 2-13: Empleo de dos redes en paralelo [75]

Uso de *3D-CNN* redes convolucionales tridimensionales

Tran y colaboradores [78] proponen el empleo de redes convolucionales tridimensionales para aplicar de manera análoga las ventajas de las redes convolucionales en el campo de análisis de vídeo (Figura 2-14). Mientras que tiene resultados prometedores con redes de baja profundidad y por lo tanto imágenes de baja resolución, es muy sensible al aumento de resolución de secuencias de entrada –ya que ahora debe tener en cuenta la dimensión temporal-, necesitando mucha potencia de procesamiento.

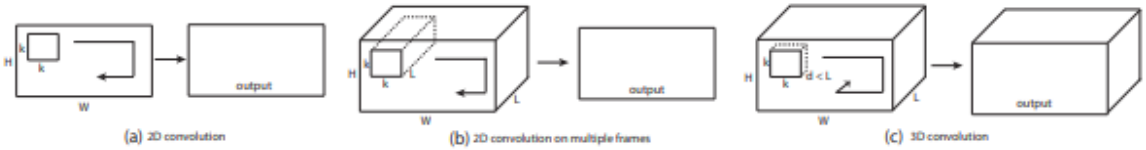

Figura 2-14: Empleo de redes convolucionales tridimensionales [78]

Empleo de *CNN* como extractor de características y *LSTM* como clasificador

Sudhakaran y Lanz (2017) [79] proponen dividir el trabajo de procesamiento en dos módulos diferenciados: una *CNN* que analice de manera individual todos los fotogramas de la secuencia, y una *LSTM* que relaciona los resultados de cada *CNN* para sacar resultado global de la secuencia (Figura 2-15).

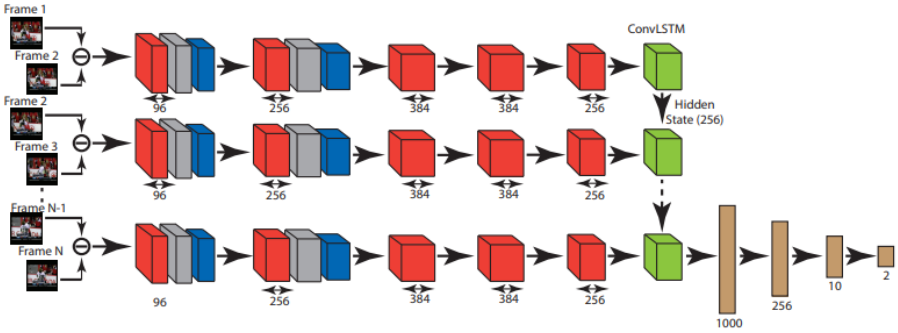

Figura 2-15: Empleo de módulos diferenciados [79]

2.4 Sistemas de detección de escenas de violencia

En este proyecto se ha decidido abordar el complejo problema de identificación de actos de violencia física en una secuencia de vídeo. Dicha acción violenta, provoca una interacción entre dos o más implicados, en la que se intenta provocar dolencia, en al menos uno de los implicados.

Se ha seleccionado este tema porque es suficientemente complicado como para poder abordarlo mediante métodos tradicionales (ya que permitirían el reconocimiento de elementos en movimiento, pero no detectarían información contextual importante para etiquetar correctamente la escena; por ejemplo, un sistema tradicional es capaz de detectar

un flujo de movimiento de un video, pero no puede distinguir qué acción ha provocado este movimiento. Sin embargo, el uso de *Machine Learning* avanzado sí lo permitiría.

Además, esta incursión abre variados campos de investigación, ya que la aplicación de los resultados a casos de vigilancia o seguridad vial son más que evidentes. Prescindir o racionalizar de alguna forma la subjetividad humana, presente en sistemas operados y certificados por el hombre, permite en conjunto racionalizar la productividad y seguridad. Otro factor importante a la hora de poder usar en la práctica un detector de escenas de violencia es el tiempo necesario para poder hacer la detección, con el objetivo de que se pueda reaccionar a esta de manera reactiva en un tiempo apropiado.

Como se analiza en el apartado número 4, la identificación de violencia es un campo complejo, porque comparte características con otras acciones claramente no violentas (rápidos movimientos que se hallan, por ejemplo, en cualquier deporte o la interacción entre 2 personas se encuentran en un saludo o un abrazo), por lo que es muy importante definir en qué puntos difiere la violencia de los anteriores comportamientos humanos. Así, los resultados del sistema propuesto se deberán valorar para comprender en qué puntos puede diferir su criterio de la definición de violencia antes comentada.

Mientras que los trabajos iniciales en este campo pusieron su atención en la detección de objetos altamente relevantes (armas, sangre, explosiones, etc.) en vez de detectar la acción violenta, Nievas y colaboradores [10] crearon los primeros *Datasets* donde se clasificaban las acciones violentas, sentando las bases del problema y facilitando así su ulterior estudio.

Para el reconocimiento directo de acción humana y detección de violencia en vídeos, se encuentran dos tipos distintos de categorías en estos algoritmos [95]: extracción de características y el uso de un clasificador (*shallow classifier*) y un sistema *end-to-end* de aprendizaje profundo. En el primer tipo de algoritmos se crean conjuntos de descriptores de vídeo y se utilizan como entrada de un clasificador lineal (*SVM*, por ejemplo). En el segundo tipo se utilizan diferentes arquitecturas de red para hacer la labor de extracción de características y clasificación, como las vistas en el punto 2.3.6.

Los mejores resultados que se han visto según distintos artículos [80, 81, 82, 83] se han obtenido usando técnicas de aprendizaje profundo, por lo que se utilizarán para este trabajo.

2.5 Conclusiones obtenidas del estudio del estado del arte

Entre todos los modelos antes descritos se ha decidido usar para este TFG la arquitectura ***CNN* como extractor de características y *LSTM* como clasificador** por las siguientes razones:

- La separación de funciones entre la *CNN* y *LSTM* permite un trabajo racional y especializado (por separado) si se compara con un sistema que combine todo en uno, ya que la *CNN* por su parte, extrae de manera sistemática las características de cada imagen, mientras que la *LSTM* se encarga de analizar el flujo espaciotemporal.
- La capacidad de aprender las características óptimas para la clasificación, que poseen las redes neuronales artificiales, las hace indispensables para el desarrollo de un modelo de detección automática.
- Las ventajas que poseen las redes convolucionales respecto a otros tipos de redes a la hora de analizar imágenes. Las redes convolucionales se han contrastado en distintas implementaciones para ser utilizadas en clasificación.
- La eficiencia computacional que aporta este diseño frente a los otros propuestos, debido a que el tiempo es sin lugar a duda un factor determinante en este problema.

3 Desarrollo del proyecto

3.1 Organización y diseño del sistema

Para este proyecto se han diseñado 2 módulos principales (Figura 3-1):

- **Módulo de entrenamiento:** contiene el diseño, la creación y el proceso de entrenamiento del modelo convolucional profundo original. En este módulo se graban los modelos para analizarlos posteriormente.
- **Módulo de análisis:** en este módulo se importan los modelos ya creados en el módulo anterior, y se extrae información cualitativa y cuantitativa sobre los resultados del entrenamiento, permitiendo analizar sus fortalezas y debilidades. Dentro de este módulo se ha realizado una implementación de mapas de calor en imagen [84] denominada *CAM,, Class Activation Mapping*, que se ha modificado para que pueda ser utilizada en vídeo.

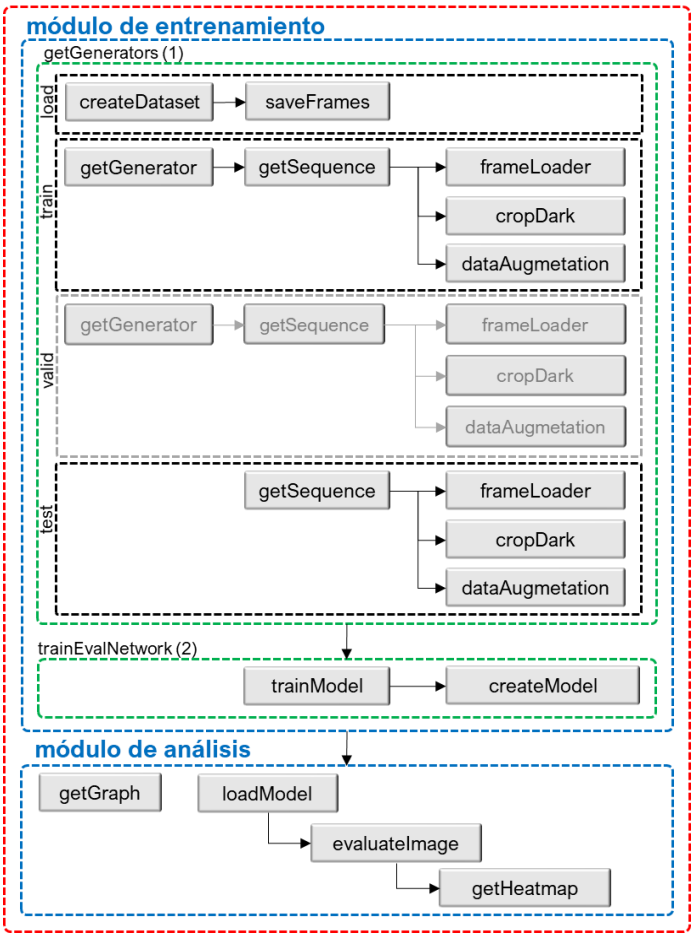

Figura 3-1: Esquema general de la organización de sistema

La mayor parte del TFG se centra en el módulo de entrenamiento, debido a que es aquí donde reside la mayor parte de la labor de investigación y desarrollo. El segundo módulo cobrará atención en el capítulo número 4 de la memoria, donde se discutirán los resultados del entrenamiento. A continuación, se introduce el modelo convolucional utilizado, el diseño y la arquitectura base, pasando por una breve explicación de sus componentes, así como el proceso de carga de imágenes necesario. Como el objetivo es entrenar una red neuronal a base de imágenes de violencia y después comprobar su eficacia, se debe

empezar por plantear un modelo que tendrá como entrada las secuencias de píxeles que forman el vídeo y que tendrá como salida la predicción deseada. Para pasar de los archivos de origen a estas secuencias de píxeles se requerirá de un módulo que lea todo el conjunto de imágenes y las transforme a un formato adecuado para el modelo. Además, debido a la necesidad de dividir el conjunto de imágenes con el que se va a hacer el tratamiento en conjuntos de entrenamiento, validación y prueba, se deben tratar estos conjuntos en su totalidad, antes de importarlos al modelo.

En el modelo original se parte de un vídeo {[20,244,244,3]} como una secuencia de imágenes de dimensiones (244×244), en la que es conveniente dividir el color de la imagen en las 3 respectivas capas (RGB) y que se tratará con un número concreto de fotogramas (20). Esta secuencia de imágenes constituye los datos de entrada de una red convolucional, que conserva a la salida la información temporal al aplicarse a cada fotograma por separado. Después se aplicará otra capa convolucional que entrelazará la información temporal entre los fotogramas y, finalmente, se usarán capas totalmente conectadas, para condensar la información extraída de las capas anteriores, con el fin de proporcionar una única salida.

3.2 Redes residuales, ResNet

El modelo seleccionado para solucionar este problema parte de *ResNet50v2*. *ResNet* es un diseño de red residual generalista, centrado en el aprendizaje de características. Las redes neuronales residuales hacen esto utilizando conexiones de omisión o atajos para saltar sobre algunas capas. Los modelos típicos de *ResNet* se implementan con saltos de doble o triple capa que contienen no linealidades *(ReLU*) y *Batch Normalization*, la misma estructura de *f (x)* explicada (4).

Además, se propuso añadir un nivel mayor de complejidad, que consiste en controlar mediante otra función *T* (con sus propios pesos), en qué casos debe influir el atajo en la salida:

$y = H(x,wh) + x$ *(función original)* (5)

Adición: Función *T(x,wt)* para regular cada una de las contribuciones mediante una distribución de Bernioulli:

$y = H(x,wh) \cdot T(x,wt) + x \cdot (1-T(x,wt))$ *(función modificada)* (6)

Esto se denomina *HighwayNet* y aporta un mayor control a la red sobre su tasa de atajos en sus capas. Por último, existen redes que poseen atajos de una capa al resto de capas posteriores, haciendo lo planteado anteriormente de manera extrema, se denominan ”redes densas” o *DenseNets* (Figura 3-2) [85].

Una motivación para saltar sobre las capas es evitar el problema del desvanecimiento de los gradientes (*vanishing gradients*), reutilizando las activaciones de una capa anterior hasta que la capa adyacente aprenda sus pesos. Durante el entrenamiento, los pesos se adaptan para silenciar la capa superior y amplificar la capa previamente omitida. En el caso más simple, solo se adaptan los pesos para la conexión de la capa adyacente, sin pesos explícitos para la capa superior. Esto funciona mejor cuando se pasa una sola capa no lineal, o cuando las capas intermedias son todas lineales. De lo contrario, se debe aprender una matriz de peso explícita para la conexión omitida (se debe usar una *HighwayNet*).

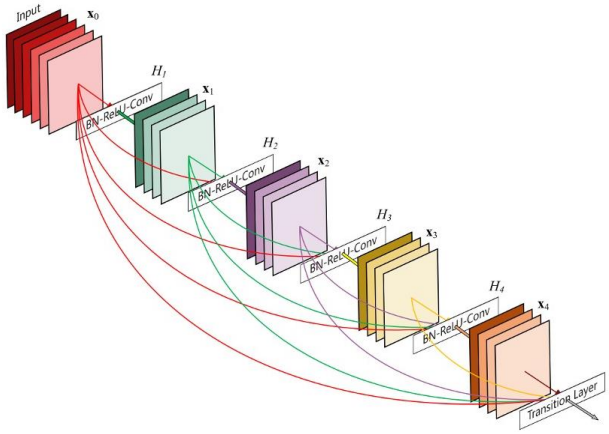

Figura 3-2: Ejemplo de red densa, atajos entre capas [85]

Saltar u omitir, efectivamente simplifica la red, usando menos capas en las etapas iniciales de entrenamiento. Esto acelera el aprendizaje al reducir el impacto de los gradientes que desaparecen, ya que hay menos capas para propagarse. Luego, la red restaura gradualmente las capas omitidas a medida que aprende el espacio de características. Hacia el final del entrenamiento, cuando todas las capas se expanden, se mantiene más cerca del colector y, por lo tanto, aprende más rápido. Una red neuronal sin partes residuales explora más del espacio de características. Esto lo hace más vulnerable a las perturbaciones que hacen que abandone el colector y necesita datos de entrenamiento adicionales para recuperarse.

Debido a las características de las redes *LSTM* y *ResNet*, se decidió utilizar esta arquitectura para la aplicación objeto de estudio, debido a que era necesario un nivel de complejidad superior a la hora de analizar secuencias de vídeo en comparación con imágenes. *ResNet50v2* es una versión [86] de *ResNet* que tiene 50 capas de profundidad. Puede cargar una versión preentrenada de la red entrenada en más de un millón de imágenes de la base de datos *ImageNet.* La red preentrenada puede clasificar las imágenes en 1000 categorías de objetos. Como resultado, la red ha aprendido representaciones de características ricas para una amplia gama de imágenes. La red tiene un tamaño de entrada de imagen de 224x 224.

Resumiendo, se ha optado por utilizar *ResNet50v2* como *RNN*, seguido de una *LSTM* para aportar la información temporal, continuando con capas totalmente conectadas hasta tener un único valor de salida. En el capítulo 4 se explicará la implementación y experimentos con diversos ajustes y parámetros.

3.3 Módulo de entrenamiento del modelo

En este apartado se detallará la estructura del módulo de entrenamiento del modelo. Está dividido en 2 secciones claras:

1. ***getGenerators*:** se hace un tratamiento inicial al conjunto de datos de entrenamiento, preparando los vídeos para su ingesta en el modelo mediante su manipulación (no muy compleja debido la utilización de *ANN*).
2. ***trainevalNetwork***: se entrena el modelo diseñado anteriormente, con los datos, y se guarda el histórico de entrenamiento, junto al modelo entrenado.

En la primera sección ***getGenerators*** se parte de un conjunto de vídeos archivados en una carpeta predeterminada donde se establecen: el título del vídeo y la categoría a la que pertenece. Como parte del procedimiento inicial se analiza el contenido la carpeta y se almacenan en una lista, automáticamente todos los vídeos encontrados. Después, se procede a la extracción de los fotogramas de estos vídeos. La cantidad de fotogramas que

se utilice para el vídeo depende del conjunto de datos o *Dataset* empleado, ya que, en función de tipo de *Dataset*, es probable que interese más un número de fotogramas u otro. Una vez extraídos los fotogramas de los vídeos, se procede a hacer una separación del conjunto de datos en: entrenamiento, validación y prueba (Figura 3-3).

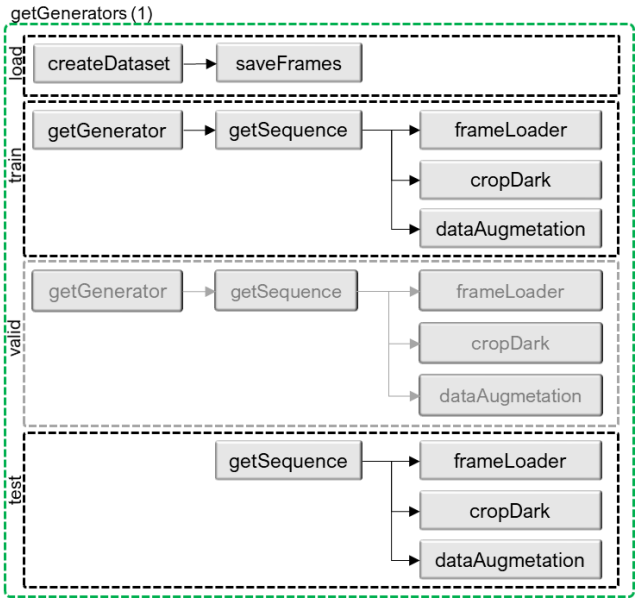

Figura 3-3: *getGenerators:* tratamiento inicial de datos

Una vez realizada esta separación, se importa la secuencia de vídeo dentro del programa y se prepara para entrenar el modelo. La importación de la secuencia de vídeo se ha implementado mediante un generador (*generator*) en *Python.* Un generador [87] es un tipo especial de función que permite recorrer una estructura iterable (como una lista), obteniendo un elemento de la lista cada vez que se llame a la función. Son muy eficaces a la hora de procesar información muy pesada, cargando solamente lo necesario en memoria. Por otra parte, se ha demostrado que el empleo de la "imagen diferencia" (obtenida entre dos fotogramas contiguos de una secuencia de vídeo) mejora el rendimiento del modelo [79].

No obstante, esto dificulta hacer un análisis coherente de los resultados, sobre todo en una implementación de mapas de calor que se hará en el apartado 4.5.

En la segunda sección ***trainEvalNetwork (2)*** se procede a obtener secuencias de vídeos de entrenamiento y validación, a partir de los generadores que se han preparado en el módulo anterior (Figura 3-4). Con estos se entrenará durante un número de lapsos, denominados "épocas" (*epochs*) con el fin, de que el modelo vaya aprendiendo sucesivamente las características de la imagen. Para garantizar la correcta comprensión del tipo de ciclo a que se refiere es necesario establecer la diferencia entre una época y una iteración. Una iteración es todo el procesado que realiza la red de un lote de datos en la fase de entrenamiento. Eso implica realizar el paso hacia delante (*forward propagation*) y el paso hacia atrás (*back propagation*). Por otra parte, una época se refiere al procesado del conjunto completo de entrenamiento en lotes de datos (iteraciones). Entonces, cada vez que el algoritmo ha procesado todas las muestras en el conjunto de datos, se ha completado una época. Se irá comprobando el resultado del entrenamiento de la red, mediante la evaluación del modelo en el conjunto de prueba y se aplicarán automáticamente los algoritmos de *backpropagation*, para ir mejorando en la precisión del modelo. En cada época se comprueba si el modelo ha mejorado con respecto al mejor modelo que hubiera guardado, y si lo ha hecho se guarda. Finalmente, una vez se llega a la iteración final, se

guarda en un archivo el histórico de entrenamientos (como ha ido variando la tasa de acierto por *epoch*, cual ha sido la tasa de error en función del conjunto de entrenamiento). En el capítulo número 4 se discuten los resultados.

Figura 3-4: *trainEvalNetwork*, sección de entrenamiento, validación y prueba

A continuación, se muestra un fragmento del código usado para cargar los fotogramas creados. A estos fotogramas se les ajusta en tamaño mediante interpolación bilineal, de forma tal que cuadren con el tamaño de figura (244x244) que requerirá el modelo, se hace un cambio de uint_8 ([0,1]) a float32 ([0,255]). Después se normalizan y escalan los valores de las imágenes a la media y la desviación estándar de las imágenes del *Dataset* de *ImageNe*t, ya que el modelo se inicializará usando esos pesos y la normalización permite controlar el dominio de los posibles valores de salida para que tenga un comportamiento apropiado.

```
def frameLoader(frames,figure_shape,to_norm = True):
  outputFrames = []
  for frame in frames:
    image = load_img(frame, target_size=(figure_shape,figure_shape),interpolation
='bilinear')
    image_array = img_to_array(image)
    figure = (image_array / 255.).astype(np.float32)
    mean = [0.485, 0.456, 0.406]
    std = [0.229, 0.224, 0.225]
    figure = (figure - mean) / std
    outputFrames.append(figure)
  return outputFrames
```

El resto de funciones (código) que componen los módulos de entrenamiento y de análisis se encuentran en los Anexos.

3.4 Técnicas de mejora de aprendizaje del modelo

Mientras que en el apartado 2.3.4 se ha explicado cuales son las principales capas del modelo, en este punto del trabajo se explicarán otras técnicas estándares utilizadas en aplicaciones de redes neuronales que ayudan al entrenamiento del modelo en diferentes formas.

Aumento de datos *(data augmentation)* [88]: generalmente se emplea para evitar sobreaprendizaje (*overlearning*) en la fase de entrenamiento del modelo (Figura 3-5). Consiste en una manipulación de las imágenes originales mediante traslación, rotación u otras técnicas de edición de imagen con el fin de recrear un dato distinto pero posible dentro del espectro de posibilidades que se quiere que cubra el modelo, sin necesitar un mayor conjunto de datos. Es de gran utilidad cuando se tiene un conjunto de datos no muy elevado y es esencial cubrir todas las casuísticas posibles con poca información disponible, o no se quiere que el modelo aprenda ciertos patrones no aplicables al caso real.

En este proyecto, se ha implementado una manipulación básica de imágenes, donde un porcentaje de las imágenes son manejadas mediante una traslación y una rotación. Cuando la resolución de la imagen es baja, los resultados con *zoom* provocan una caída de calidad que distorsiona sustancialmente la imagen.

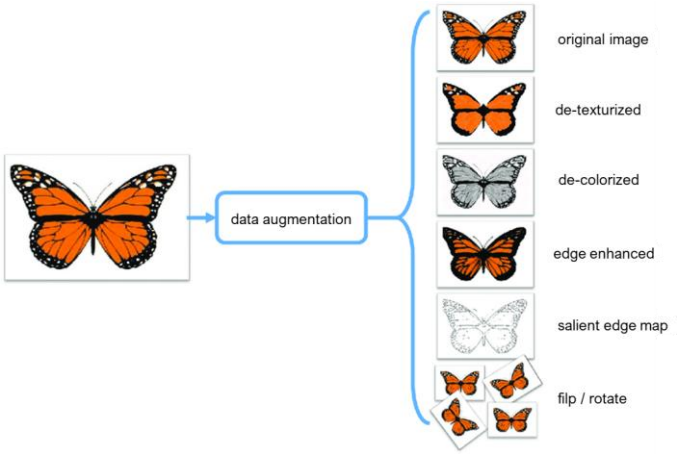

Figura 3-5: Esquema de ejemplo de aplicación de *data augmentation* [88]

Regularizadores de *kernel* [5]: como se ha explicado, la forma en la que las redes neuronales entrenan es mediante la modificación sistemática de los pesos de cada una de las neuronas de sus redes entrenables. Esta modificación se realiza con un algoritmo de propagación inversa, *backpropagation*, que modifica mediante la aproximación de descenso por gradiente los pesos de cada capa desde la capa de salida (que hemos comprobado su resultado debido a ser un problema de aprendizaje supervisado) hasta las capas iniciales. Existe un elemento llamado regularizador que se encarga de aplicar penalizaciones a parámetros de entrenamiento durante este proceso de optimización, aplicados por capa (Figura 3-6). Esta penalización consiste en disminuir el peso de aquellos con valor absoluto alto, con lo cual la regularización tiende a llevar a cero los pesos que no son críticos para el funcionamiento de la red.

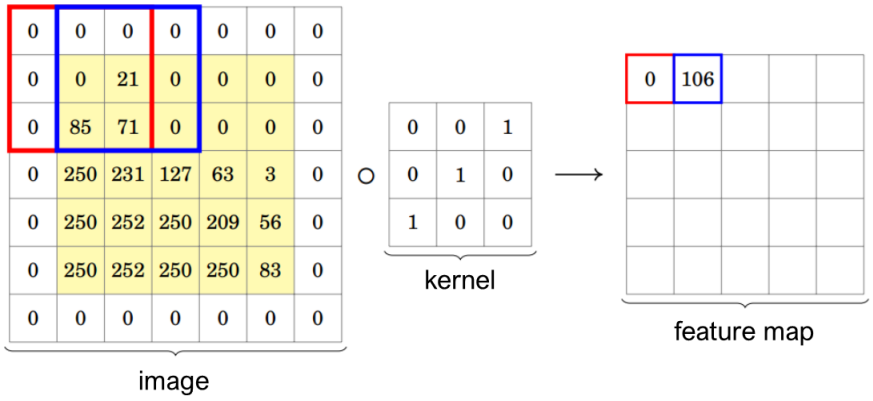

Figura 3-6: Esquema de ejemplo de aplicación del regularizador de *kernel* [5]

Como también se ha explicado, existen regularizadores de *kernel*, de *bias* y de salida u *output*. En este caso se emplean regularizadores de *kernel* exclusivamente. Estos regularizadores tienen varios tipos, ya sea *L1* (que penaliza basándose en el valor absoluto del parámetro) o *L2* (que penaliza basándose en el cuadrado del parámetro). Existe la posibilidad de utilizar ambos a la vez, denominado regularizador *L1L2*.

Dropout [89]: otro procedimiento clásico que se aplica a este modelo es el uso de *dropout*, o pérdidas intencionadas de información entre capas del modelo (Figura 3-7). Este procedimiento se usa para generar un factor de "pérdida controlada de información" dentro del propio modelo, lo que provoca que no se pueda influenciar de manera continuada por elementos poco significativos de la imagen que puedan aparecer en pocos aspectos, sino que se fijará en las características que aparecen de manera general.

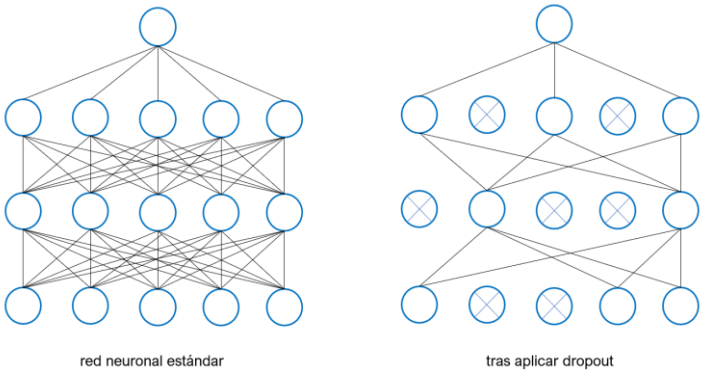

Figura 3-7: Esquema del modelo de pérdidas intencionadas [89]

3.5 Módulo de análisis

En este módulo se importa el modelo anteriormente entrenado y se le introducen el o los vídeos que interesa evaluar según el modelo (Figura 3-8). El modelo devolverá una puntuación entre 0 y 1, debido a la utilización de una sigmoide como función de activación final, lo cual permitirá discutir cuán seguro o no está el modelo de la existencia de violencia en el vídeo.

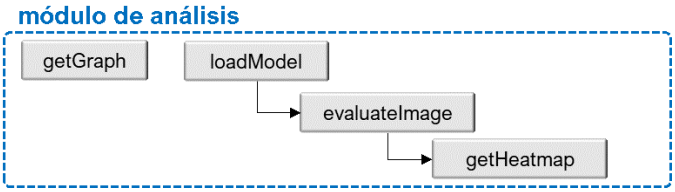

Figura 3-8: Esquema del módulo de análisis

Finalmente, se realiza una implementación de mapas de calor, que permite identificar cuáles son las zonas del vídeo a las que el modelo ha prestado atención y así poder inferir la presencia de violencia o falta de ella (mayor detalle sobre este tema en el apartado 4.3).

3.6 Integración

El lenguaje en el que se ha desarrollado la implementación es *Python*, usando *Keras* [90]. *Keras* es un *framework* de código abierto desarrollado para aprendizaje profundo caracterizado por simplificar el proceso de creación y mantenimiento de modelos que tiene por debajo *TensorFlow* [91], que es una biblioteca también de código abierto de Google. *Keras* ofrece como bibliotecas alterativas *Theano* [92] o *PyTorch* [93]

Se ha implementado sobre *Google Colaboratory*, un sistema en línea igualmente proporcionado por Google que permite acceso a una sesión de *Python*, de manera gratuita y gran capacidad de cómputo (Figura 3-9).

Debido a la gran necesidad de cómputo que requieren las redes neuronales, entrelazadas con redes recurrentes como *ResNet*, se opta por esta alternativa, ya que *Google Colaboratory* ofrece de manera nativa compatibilidad con *TensorFlow* (ya que son sus creadores).

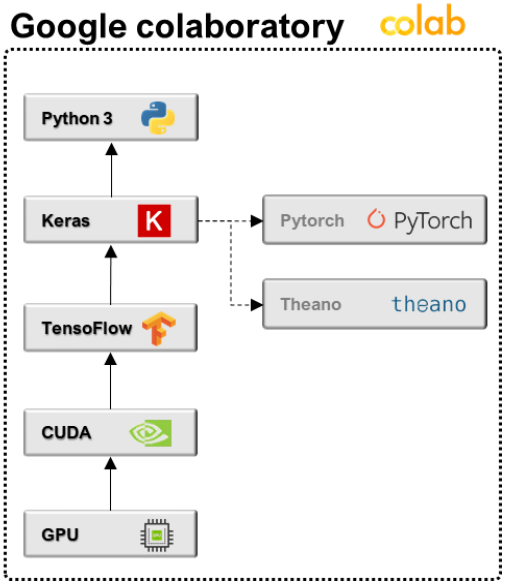

Figura 3-9: Esquema de integración soportado por *Google Colaboratory*

3.7 Datasets utilizados

Se han investigado un conjunto de *Datasets* (Tabla 2) en función del contenido, escenario y escala, esto ha permitido elegir un subconjunto de ellos ser utilizados en el estudio:

Tabla 2. Conjunto de *Dataset* analizados

Dataset	**Autor / año**	**Escala**	**Duración/ secuencia (s)**	**Resolución**	**Etiquetado**	**Escenario**
BEHAVE	Blunsden y colaboradores. 2009 [94]	4 vídeos (171 secuencias)	0,24-61,92	640×480	nivel-fotograma	peleas actuadas
RE-DID	Rota y colaboradores. 2015 [95]	30 vídeos	20-240	1280×72	nivel-fotograma	natural
VSD	Demarty y colaboradores. 2015 [96]	18 películas (1317 secuencias)	55,3-829,4	variable	nivel-fotograma	película
CCTV-Fights	Perez y colaboradores. 2019 [97]	1000 secuencias	5-720	variable	nivel-fotograma	natural
HockeyFights	Nievas y colaboradores. 2011 [98]	1000 secuencias	1,6-196	360×288	nivel-vídeo	partidos de Hockey
MoviesFights	Nievas y colaboradores. 2011 [99]	200 secuencias	1,6-2	720×480	nivel-vídeo	película
ViolentFlows	Hassner y colaboradores. 2012 [100]	246 secuencias	1,04-6,52	variable	nivel-vídeo	natural
SBU Kinect Interaction	Yun y colaboradores. 2012 [101]	264 secuencias	0,67-3	640×480	nivel-vídeo	peleas actuadas
UCF-Crime	Sultani y colaboradores. 2019 [102]	1900 secuencias	60-600	variable	nivel-vídeo	vigilancia
RWF-2000	Cheng y colaboradores. 2019 [103]	2000 secuencias	5	variable	nivel-vídeo	vigilancia

Se han elegido para trabajar, *Datasets* etiquetados a nivel-vídeo, que tienen una variedad suficiente de escenarios, lo que permite comprobar la eficacia del sistema. Finalmente se han seleccionado 4 *Datasets* de los 6 disponibles con las características referidas: *HockeyFight*[A], *MoviesFight*[B], *ViolentFlows*[C] (también conocido como *CrowdViolence*) y *RWF-2000*[D], este último de publicación muy reciente. *UCF-Crime* se presentaba como un *Dataset* interesante, sin embargo, la duración de las secuencias es muy elevada, en interés del trabajo académico, resulta atinado analizar intervalos cortos de vídeo para conseguir resultados comparativos, con una menor cantidad de procesamiento, luego *UCF-Crime* se descartó. *SBU Kinect Interaction* también se ha descartado, en favor de *MoviesFight*, ya

que también posee escenas actuadas de violencia, y su licencia de uso -con interés académico- está disponible.

Se ha creado un modelo entrenado con un conjunto de entrenamiento para cada uno de los cuatro citados *Datasets* y se ha programado automáticamente la evaluación del modelo de entrenamiento y prueba para comprobar la eficacia del modelo, terminando con un análisis de los parámetros de entrenamiento (tasa de acierto de entrenamiento, validación y prueba) que permite evaluar la eficacia del entrenamiento sobre el modelo. Para finalizar, se ha hecho un análisis de los resultados revisando el contenido el vídeo, discutiendo posibles fallos del sistema. Se implementa, además, una solución para intentar comprender situaciones en las que el sistema no detecta bien características de la imagen. En estas pruebas, los modelos se han entrenado un total de 30 épocas *(epoch)*, mejorando progresivamente su tasa de acierto. Se ha analizado su comportamiento en el conjunto de entrenamiento contra el conjunto de validación para comprobar que no haya sobreajuste o infraajuste y, mediante los resultados se ha comprobado el correcto funcionamiento del modelo. Mientras que se ha usado *ResNet50v2* como *CNN*, se ha experimentado con *ResNet152v2* (*ResNetv2* con 152 niveles de profundidad, Inceptionv3 (que plantea distintas resoluciones de convolución en paralelo en vez de uso de residuos [104]), *Incepción-ResNet* (un híbrido [105] que combina las 2 soluciones de *ResNet* e *Inception*). Se explicarán los resultados de los distintos modelos en el apartado 4.2.

A continuación, se presenta un ejemplo de funcionamiento del modelo para cada uno de los *Datasets*.

1.- *HockeyFights*, este *Dataset* fue recopilado de partidos de la Liga Nacional de Hockey (EE. UU.) y fue diseñado para juzgar los vídeos como violentos o normales. *HockeyFights* contiene específicamente 1000 vídeos con una resolución de 360×228 píxeles, que se dividen en cinco divisiones para cada categoría (Figura 3-10). Tiene como particularidad que, en las secuencias de vídeo de los partidos, se presenta la problemática de un fondo uniforme y elementos distinguidos (jugadores contra campo y público) y la dificultad de analizar acciones deportivas y marcar un límite entre acciones violentas y las no violentas.

Nombre vídeo: no80_xvid.avi
Descripción: Entrada a portería
Categoría (*label*): No violencia

Fotograma 1 Fotograma 5 Fotograma 15

Figura 3-10: Secuencias de vídeo de un partido de Hockey

2.- *MoviesFight* (Figura 3-11) es un *Dataset* más variado. Contiene 200 escenas extraídas de películas cortas, tanto de violencia como de no violencia. En una primera aproximación, se puede entender que el modelo pueda distinguir bien los casos de violencia, ya que las escenas de acción y consecuentemente violentas, poseen mucho más movimiento y desplazamientos bruscos de cámara que escenas no violentas.

Nombre vídeo: 90.mpg
Descripción: persona caminando
Categoría (*label*): No violencia

Fotograma 1 Fotograma 5 Fotograma 15

Figura 3-11: Secuencia de vídeo de persona caminando en *MoviesFight*

3.- *ViolentFlows* (Figura 3-12) es una base de datos de secuencias de vídeo del mundo real sobre violencia colectiva, junto con protocolos de referencia estándar diseñados para evaluar tanto la clasificación violenta / no violenta como las detecciones de brotes de violencia. El conjunto de datos contiene 246 vídeos. Todos los vídeos fueron descargados de YouTube. La duración más corta del clip es de 1,04 s, la más larga es de 6,52 s. La duración promedio de un vídeo es de 3,60 s presenta el caso de multitudes, tanto violentas como no violentas. Destaca esta gran diferencia con respecto al resto de *Datasets*, donde por lo general los enfrentamientos se sucedían entre 2 o un número bajo de personas. El caso clásico de violencia que se aprecia en este *Dataset* son peleas de bandas callejeras o *hooligans* de partidos de fútbol americano.

Nombre vídeo: Hooligans_violence_English_Hooligans__JohnLaw_Wob3r1Leamw.avi
Descripción: pelea entre hooligans
Categoría (*label*): Violencia

Fotograma 1 Fotograma 5 Fotograma 15

Figura 3-12: Secuencia de violencia en *ViolentFlows*

4.- *RWF-2000* (Figura 3-13) es una base de datos de secuencias con 2.000 vídeos capturados por cámaras de vigilancia en escenas del mundo real, en distintas situaciones cotidianas. El problema característico que se presenta en este *Dataset* es la larga duración de las secuencias y el formato de vídeo en 16:9 en vez de 4:3, en comparación al resto de los anteriores *Datasets*. De todas maneras el modelo trabajará con fotogramas de 244×244 píxeles, por lo que se presentará distorsión de la imagen a la hora de analizarla.

Nombre vídeo: FIGHT_0Ow4cotKOuw_2.avi
Descripción: pelea en un bar
Categoría (*label*): Violencia

Fotograma 1 Fotograma 5 Fotograma 15

Figura 3-13: Secuencia de vídeo con violencia *Dataset RWF-2000*

3.8 Medida de la calidad de las predicciones

Para medir la calidad de las predicciones de un algoritmo de clasificación se utiliza el denominado "informe de clasificación" o matriz de confusión [106]. Donde se resumen cuántas predicciones son verdaderas y cuántas son falsas (Figura 3-14). Específicamente, los positivos verdaderos, los falsos positivos, los verdaderos negativos y los falsos negativos, se utilizan para predecir las métricas de un informe de clasificación como se muestra a continuación. El informe muestra las principales métricas de clasificación de precisión, recuperación y puntaje F_1 por clase. Las métricas se calculan utilizando verdaderos y falsos positivos, verdaderos y falsos negativos. Positivo y negativo en este caso, son nombres genéricos para las clases predichas. Existen cuatro formas de verificar si las predicciones son correctas o incorrectas:

1. **TN / *True Negative*** (Verdadero Negativo): caso negativo y se predijo negativo.
2. **TP / *True Positive*** (Verdadero Positivo): caso positivo y se predijo positivo.
3. **FN / *False Negative*:** (Falso Negativo): caso positivo, pero se predijo negativo.
4. **FP / *False Positive*:** (Falso Positivo): caso negativo, pero se predijo positivo.

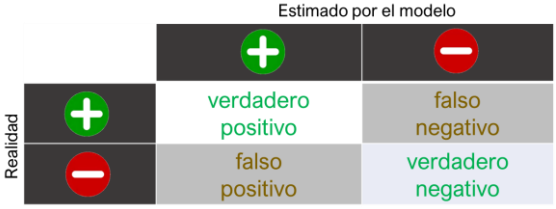

Figura 3-14: Matriz de confusión

Precision **–** ¿Qué porcentaje de las predicciones fueron correctas? *Precision* es la capacidad de un clasificador de no etiquetar una instancia positiva, que en realidad es negativa. Para cada clase, se define como la relación de positivos verdaderos, con la suma de positivos verdaderos y falsos.

TP – Verdadero Positivos *(True Positives)*
***FP* –** Falsos positivos *(False Positives)*

Precision – Precisión de predicciones positivas.

Precision = TP/(TP + FP)

Recall – ¿Qué porcentaje de los casos positivos se captó? *Recall* es la capacidad de un clasificador para encontrar todas las instancias positivas. Para cada clase, se define como la relación de verdaderos positivos, con la suma de verdaderos positivos y falsos negativos.

FN – Falsos negativos *(False Negatives)*
Recall: Fracción de positivos que fueron identificados correctamente.
Recall = TP/(TP+FN)

F_1-score – ¿Qué porcentaje de predicciones positivas fueron correctas? *F_1-score* es una media armónica ponderada de precisión y recuperación de modo que la mejor puntuación es 1 y la peor es 0. En términos generales, los puntajes de F1 son más bajos que las medidas de precisión, ya que incorporan precisión y memoria en su cálculo. Como regla general, el promedio ponderado de F_1 debe usarse para comparar modelos clasificadores, no la precisión global.

F_1-Score = 2(Recall * Precision) / (Recall + Precision)*

Otra métrica que permite establecer comparaciones consiste en calcular el **porcentaje de aciertos** del modelo para los datos contemplados. Esta métrica a pesar de ser útil en muchos problemas no tiene en cuenta el número de ejemplos de cada clase y esto impide reflejar fielmente la calidad del clasificador.

3.9 Análisis cualitativo de la imagen. Mapas de calor

Con el objetivo de ayudar al análisis de los resultados, se ha implementado un algoritmo de mapa de calor [84] (figura 3-15). Un mapa de calor consiste en una técnica de visualización de datos que muestra la magnitud de un valor como un color en un espacio de dos dimensiones. Se aplica en análisis de imagen para comprobar cómo se comporta el valor en el espacio.

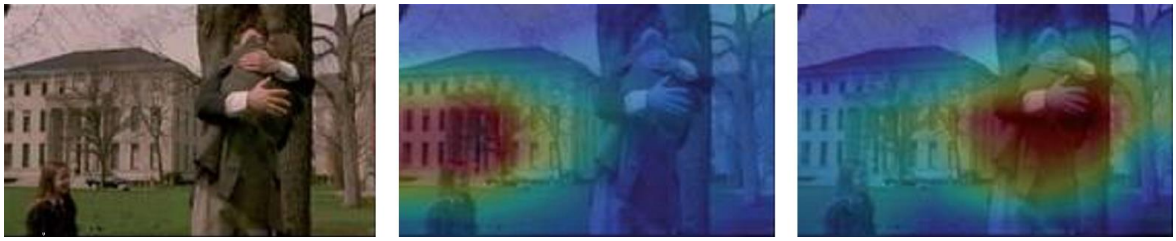

Figura 3-15: Reconocimiento de acciones en vídeo mediante mapas de calor [84]

El algoritmo llamado *Grad-CAM* [107, 108], *Gradient-weighted Class Activation Mapping*, obtiene las zonas de interés de la imagen a partir de los pesos de la capa de la red recurrente, en este caso, de *ResNet50*. La Figura 3-16 muestra un ejemplo de empleo de las *CAM, Class Activation Mapping*, generadas a partir de las 5 categorías principales predichas para la imagen dada, con la verdad básica como “domo”. La clase predicha y su puntuación se muestran en la parte superior de cada mapa de activación de clase. Se observa que las regiones resaltadas varían según las clases predichas, por ejemplo, el domo activa la parte superior redonda mientras que el palacio activa la parte plana inferior de la composición [84].

La simple modificación de la capa de agrupación promedio global, combinada con la técnica de *CAM, Class Activation Mapping* permite que la *CNN* entrenada en clasificación clasifique la imagen y localice regiones de imagen específicas de clase en un solo paso hacia adelante, para una amplia variedad de tareas, incluso aquellas para las que la red no estaba capacitada originalmente.

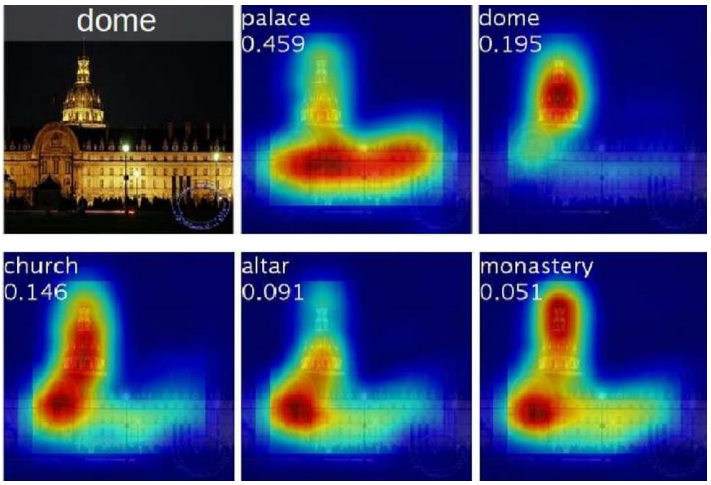

Figura 3-16: Ejemplos de *CAM* generados a partir de las 5 categorías predichas [84]

Por ejemplo, el cepillo para lavarse los dientes y la motosierra para cortar árboles (Figura 3-17) [84].

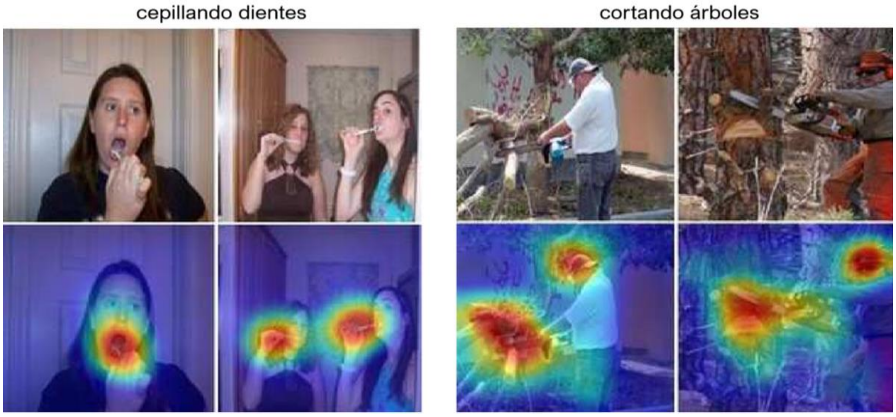

Figura 3-17: Ejemplos de clasificación y localización de regiones *CAM* [84]

En este trabajo se ha adaptado dicho algoritmo para que pueda funcionar con vídeo y permita identificar qué elementos han sido característicos para la red recurrente, para obtener los puntos clave a la hora de evaluar una imagen como violenta o no violenta.

En el desarrollo original [109], de Adrian Rosebrock de marzo de 2020, se analizan los pesos de la última capa convolucional del modelo y se obtiene una visualización en forma de mapa de calor, donde se muestran los pesos de la capa en función del número de categorías de salida. En este proyecto, únicamente se tiene una categoría de salida (violento/ no violento) por lo que un mapa de calor sencillo es suficiente.

Durante este trabajo se ha realizado una adaptación de este desarrollo para adecuarlo al modelo creado. La adaptación del modelo surge de obtener la información de los pesos de la capa *ResNet50v2* y configurar su salida para obtener un mapa de calor por cada uno de los fotogramas de la imagen. Esta implementación se puede realizar, gracias a que el modelo está diseñado para trabajar con secuencias de imágenes como entrada de *ResNet*, utilizando el contenedor *TimeDistributed* de *Keras*, que permite utilizar *ResNet* en cada uno de los fotogramas en paralelo.

4 Estudio experimental

4.1 Descripción del proceso mediante pruebas en HockeyFight

Durante las etapas iniciales del proyecto, se analizó una investigación sobre los distintos modelos convolucionales soportados por *Keras*. Como se ha explicado en capítulos anteriores, *ResNet* es un buen modelo para el problema que se quiere resolver, y está soportado de manera nativa por *Keras*. Es necesario definir entonces con qué hiperparámetros (se definen como tales a los parámetros cuyo valor controlan el ajuste, y no están limitados a los pesos del modelo, si no que incluyen otros parámetros externos) se va a entrenar el modelo. En una primera instancia, se planteó el uso de un ajuste automático de hiperparámetros (*automatic hyperparameter tuning*), pero la gran carga computacional que conlleva la combinación de *CNN+LSTM* hizo inviable este ajustador. Durante estas pruebas, no se han probado con más de 30 épocas debido a los límites de uso que establece Google Colaboratory de los recursos disponibles en la plataforma. Se ha usado HockeyFight como Dataset para obtener los hiperparámetros del modelo.

En una prueba inicial, se estableció arbitrariamente una cantidad de 20 fotogramas por vídeo para analizar, no hacer uso de aumento de datos, ni *dropout* ni *kernel regularizer*, usar una *LSTM* de 64 filtros, *ResNet50V2* usando los pesos entrenados de *ImageNet* y no reentrenable, el optimizador *Adam* y un *learning rate* de 1e-6. La Figura 4-1, muestra la gráfica de entrenamiento en *HockeyFights*.

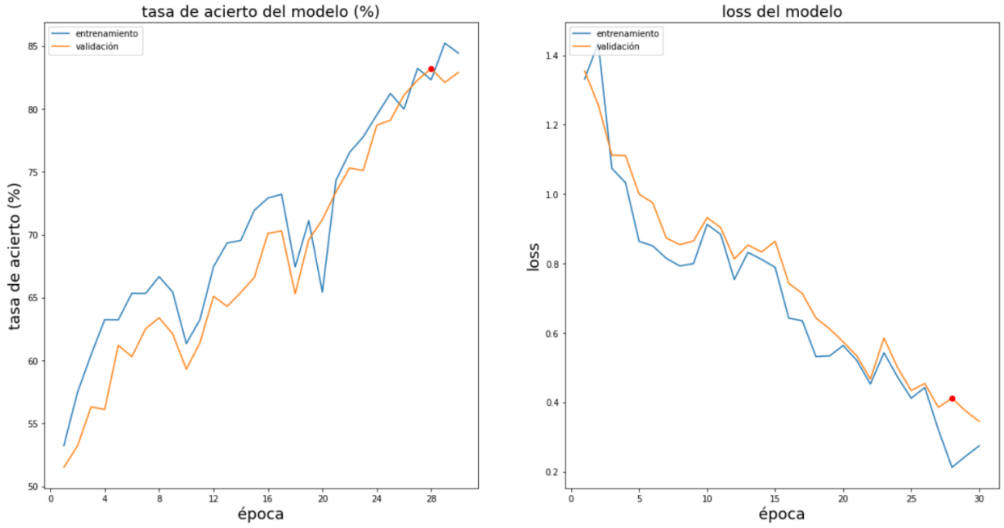

Figura 4-1: Primer modelo creado

Tiene un resultado final en validación del 83%, a priori un buen resultado, pero con presunto margen de mejora. En una primera vista a la gráfica parece que, por el momento, no se está sobreajustando. Por lo tanto, parece que no va a ser necesario introducir ajustes por esta parte, y el problema principal es que está aprendiendo de manera lenta y el *loss* es considerablemente alto. Estos síntomas, indican que se pude tratar de un problema de la tasa de aprendizaje, que es más baja de lo que debería y está tardando muchas épocas en converger.

En una segunda prueba, se ha subido el *learning rate* a 1e-5 y se ha usado *RMSprop* como nuevo optimizador, para intentar producir una convergencia más rápida. La gráfica se muestra en la Figura 4-2.

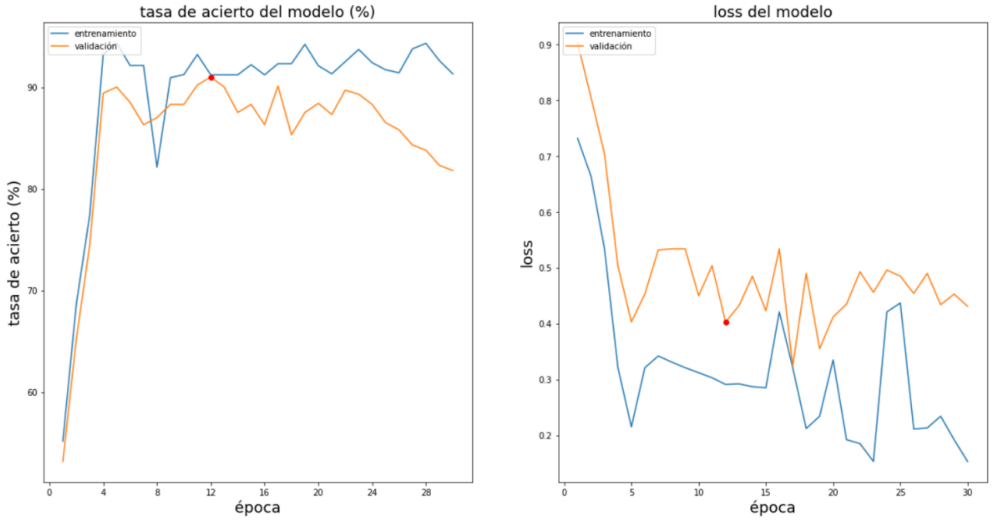

Figura 4-2: Modelo modificado en tasa de aprendizaje y optimizador

Como se puede observar, en este caso el modelo ha aprendido de manera mucho más rápida, consiguiendo tasas por encima del 90% en entrenamiento y más del 85% en validación, lo cual indica que era conveniente subir la tasa de aprendizaje. Sin embargo, los resultados, siguen sin ser tan buenos como se esperaba y a partir del *epoch* 20 se empieza a notar un claro sobreajuste del modelo debido a la bajada de la tasa de acierto en validación. Para atenuar esto, en la siguiente prueba se ha añadido un *dropout* de 0,1 entre las capas totalmente conectadas y se han aumentado un 25% de los datos, haciendo que este porcentaje tenga aleatoriamente un aumento de cámara y/o una traslación, con el objetivo de generar nueva información. Además, se baja ligeramente el *learning rate* a 5e-6, para comprobar su efecto. Los resultados se muestran en la Figura 4-3.

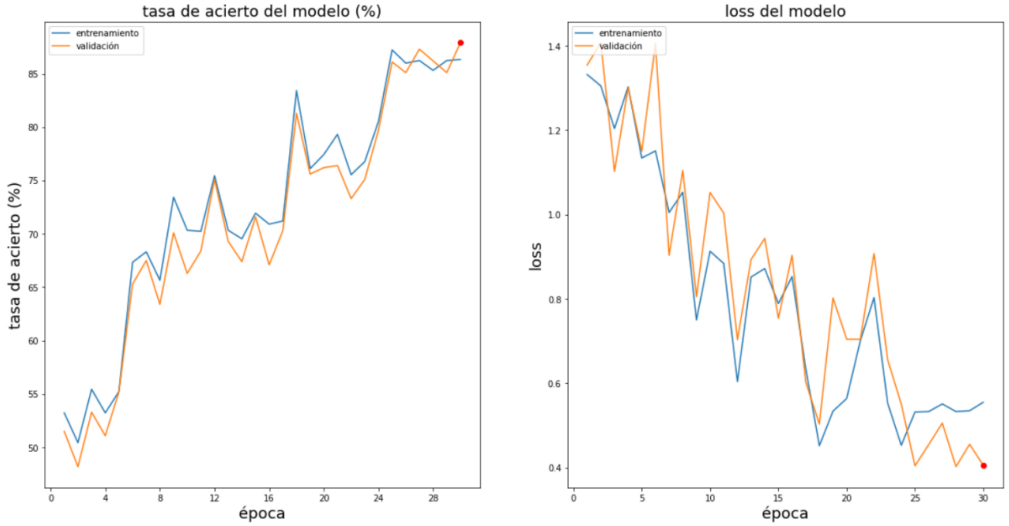

Figura 4-3: Modelo modificado con aumento de datos y *dropout*

Aparentemente el modelo ahora no sobreajusta, pero los *scores* no llegan a alcanzar en el mismo número de épocas las tasas anteriores, ya que ahora el aprendizaje es lento. Por lo tanto, incrementamos la tasa de aprendizaje para poder valorar si se produce finalmente sobreajuste o no. A partir de esta iteración, se realizaron varias pruebas en función de lo observado experimentalmente en las anteriores pruebas, y se probaron distintas combinaciones de parámetros, para optimizar la solución.

A continuación, en la Tabla 3 se muestra un resumen de las principales pruebas que se han ejecutado, con distintos parámetros, tasa de acierto, pérdida y si estaba sobreajustando *(overfitting)*.

Tabla 3. Resumen de pruebas realizadas

Fotogramas	Reentrenar	*data augmentation*	*Dropout*	*Kernel Regularizer*	Filtros de la *LSTM*	*Learning rate*	Algoritmo de optimización	Train	Test	*¿Overfitting?*
30	No	0%	0%	0	128	1e-6	*Adam*	84%	83%	No
30	No	0%	0%	0	128	1e-5	*RMSProp*	93%	85%	Si
30	Si	25%	10%	L1L2 1e-4	128	5e-6	*RMSProp*	85%	85%	No
30	Si	25%	10%	L1L2 1e-4	128	1e-5	*RMSProp*	90%	85%	Si
30	Si	50%	10%	L1L2 1e-4	128	1e-5	*RMSProp*	93%	88%	No
30	Si	50%	10%	L1L2 1e-4	128	2e-5	*RMSProp*	95%	90%	No
30	Si	75%	10%	L1L2 1e-4	128	2e-5	*RMSProp*	94%	90%	No
30	Si	75%	10%	L1L2 1e-3	128	2e-5	*RMSProp*	92%	88%	No
30	Si	75%	10%	L1L2 1e-4	128	5e-5	*RMSProp*	95%	90%	No
30	Si	50%	10%	L1L2 1e-4	128	5e-5	*Adam*	94%	90%	No
30	Si	50%	10%	L1L2 1e-4	128	5e-5	*RMSProp*	97%	92%	No
30	Si	50%	20%	L1L2 1e-4	128	5e-5	*RMSProp*	95%	90%	No
30	Si	50%	50%	L1L2 1e-4	128	5e-5	*RMSProp*	83%	79%	No
30	Si	50%	10%	L1L2 1e-3	256	5e-5	*Adam*	100%	94%	No
30	Si	50%	10%	L1L2 1e-4	256	5e-5	*Adam*	100%	95%	No
30	Si	50%	10%	L1L2 1e-4	512	5e-5	*Adam*	100%	95%	No
30	Si	50%	10%	L1L2 1e-4	256	5e-5	*RMSprop*	100%	96%	No
10	Si	50%	10%	L1L2 1e-4	256	5e-5	*RMSprop*	88%	91%	No
50	Si	50%	10%	L1L2 1e-3	256	5e-5	*RMSprop*	100%	96%	No

A partir del proceso experimental, se sacan las siguientes conclusiones de entrenamiento:

- No se nota una mejora entre usar más o menos fotogramas dentro de un mismo vídeo a priori, por lo que se usarán arbitrariamente 20 fotogramas, por establecer el tiempo de detección de violencia de una secuencia en menos de un segundo en los *Datasets* usados (que tienen una tasa de 25-30 fotogramas por segundo, dependiendo del Dataset)
- El uso de datos aumentados resultó ser necesario a partir de un *learning rate* superior a 5e-6, y se experimentó con un 25% de datos aumentados, un 50% de datos aumentados y un 75%, siendo 50% la opción más efectiva ya que mientras hubo mejoría en el paso de 25% a 50%, no la hubo de 50% a 75%.
- Junto al uso de datos aumentados se experimentó con *dropout* de entre 0,1 y 0,5, siendo 0,1 la mejor opción ya que se realiza una pérdida controlada de información suficiente para evitar sobreajuste.
- Se probaron 3 algoritmos de optimización (*Adam, SGD y RMSprop*) para el modelo y en líneas generales se comportaban de manera análoga, aunque *RMSprop* fue el más efectivo, provocando un aprendizaje más rápido.
- Se hicieron pruebas donde se permitía el reentrenamiento de los pesos de *ResNet50,* y con ellas se usó *kernel regularizer* para evitar sobreescribir completamente la inteligencia adquirida en su entrenamiento con *ImageNet*. Se utilizó el *kernel regularizer L1L2* para aprovechar, las ya descritas ventajas de la combinación, y así conseguir un mejor rendimiento.
- Se probó a entrenar el modelo sin utilizar los pesos de *ImageNet,* pero los resultados no fueron satisfactorios.
- Se probaron filtros de *LSTM* de 128, 256 y 512 filtros. Mientras que el paso de 128 a 256 filtros aportó mejora al modelo, el paso de 256 a 512 filtros tuvo resultados despreciables.
- Se estableció una cantidad de 30 *epochs* como máximo valor de entrenamiento, ya que los modelos finales convergen alrededor de este número.

A continuación, se presentan de forma resumida los parámetros seleccionados para ser utilizados en el modelo final (Tabla 4).

Tabla 4. Parámetros usados en el modelo final

Parámetro	Descripción
épocas (*epochs*)	30
data Augmentation	50% de las imágenes
dropout	0,1
kernel regularizer L1L2	0,001
LSTM de *kernel*	3×3 de 256 neuronas
ResNet50v2	con pesos de *ImageNet* y reentrenable
tasa de aprendizaje	0,00005 (para descenso por gradiente)
algoritmo de optimización	*RMSprop*

4.2 Comparación entre arquitecturas en HockeyFight

Se hace una comparación entre las distintas arquitecturas del modelo en un mismo Dataset (HockeyFight) para justificar la elección de *ResNet50* como modelo principal del proyecto.

En los siguientes gráficos se podrá apreciar cuál ha sido el histórico de aprendizaje de los modelos (Tabla 5). Se observa, por una parte, la tasa de acierto del modelo -que indica su acierto a la hora de autoevaluarse- y por otra la pérdida o *loss* del modelo, que indica "por cuánto" se ha equivocado en la predicción según la función de error, con respecto al resultado correcto.

Tabla 5. Parámetros entrenables y dilación del entrenamiento para cada modelo

Modelo	Parámetros entrenables	Dilación de entrenamiento (horas)
ResNet50v2	49.118.077	310s por *epoch* × 30 = 9.300 s/ 60/ 24 = 6,45 h
ResNet151v2	83.786.621	528s por *epoch* × 30 = 15.840 s/ 60/ 24 = 11 h
InceptionV3	45.571.486	262s por *epoch* × 30 = 7.860 s/ 60/ 24 = 5,45 h
Inception-ResNet	76.181.516	480s por *epoch* × 30 = 14.400 s/ 60/ 24 = 10 h

***ResNet50v2* en Hockey**: tiempo de aprendizaje normal, estableciendo su máxima tasa de acierto sobre el *epoch* 15 para entrenamiento y con un tiempo de entrenamiento estándar para el volumen de datos con el que se alimenta al proceso, con acierto del 96% en test (Figura 4-4).

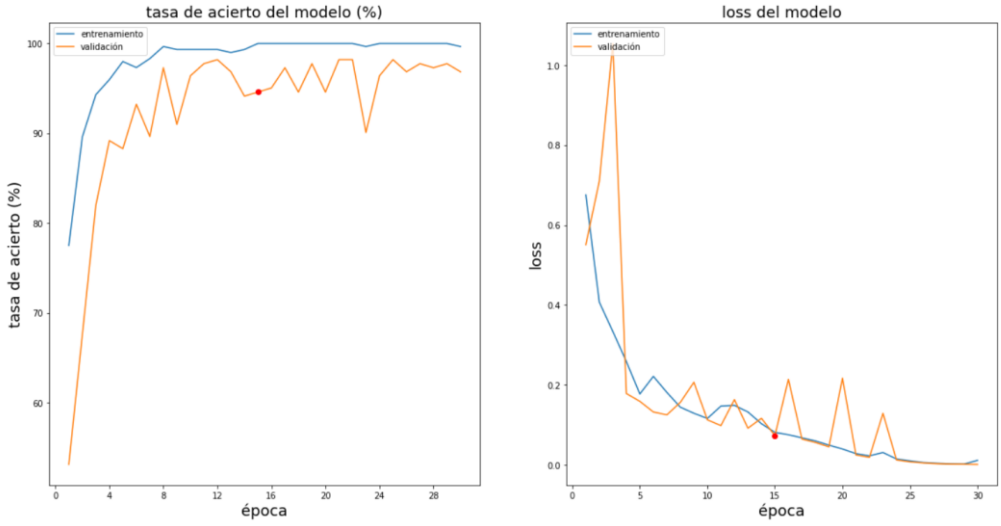

Figura 4-4: *ResNet50v2* en Hockey

***ResNet151v2* en Hockey:** tiene un comportamiento muy similar a *ResNet50*, quizá un poco más refinado, con un ligero mejor resultado en validación/test, 98%. No obstante, el tiempo de computación es mucho mayor, yéndose a las 11 h (Figura 4-5).

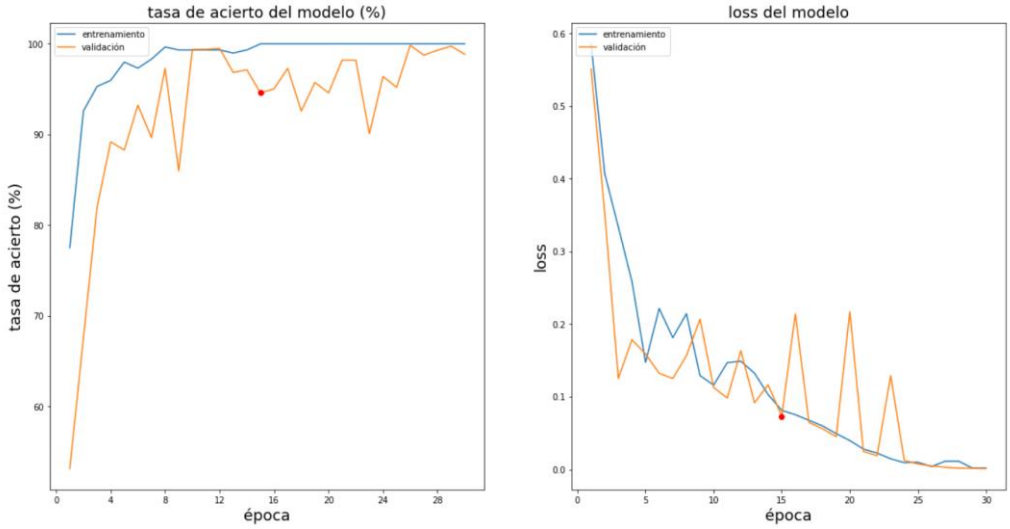

Figura 4-5: *ResNet151v2* en Hockey

***InceptionV3* en Hockey**: Se observan resultados mucho peores, apenas alcanza el 90% al *epoch* 28 y mucho mayor pérdida (Figura 4-6). El tiempo de computación es ligeramente menor que el de *ResNet50v2* (1 h menos).

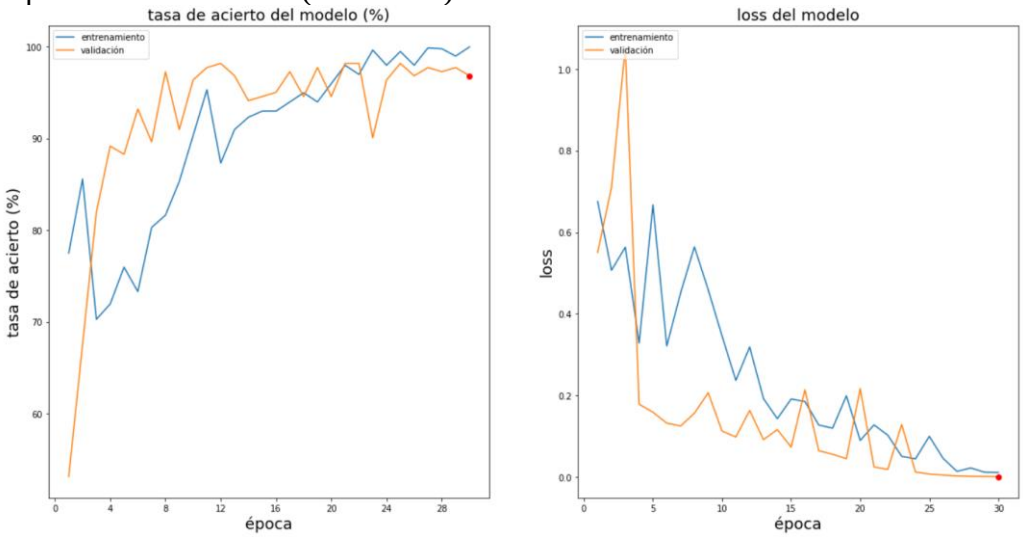

Figura 4-6: *InceptionV3* en Hockey

***Inception-ResNet* en Hockey:** es, sin duda, el modelo que mejor resuelve el problema. No obstante, compromete sobremanera la capacidad de computación (Figura 4-7). Comparado con *ResNet151v2* tarda 1 h menos y genera unos resultados iguales, si no mejores gracias a la combinación de *Inception* con *ResNet* (99%).

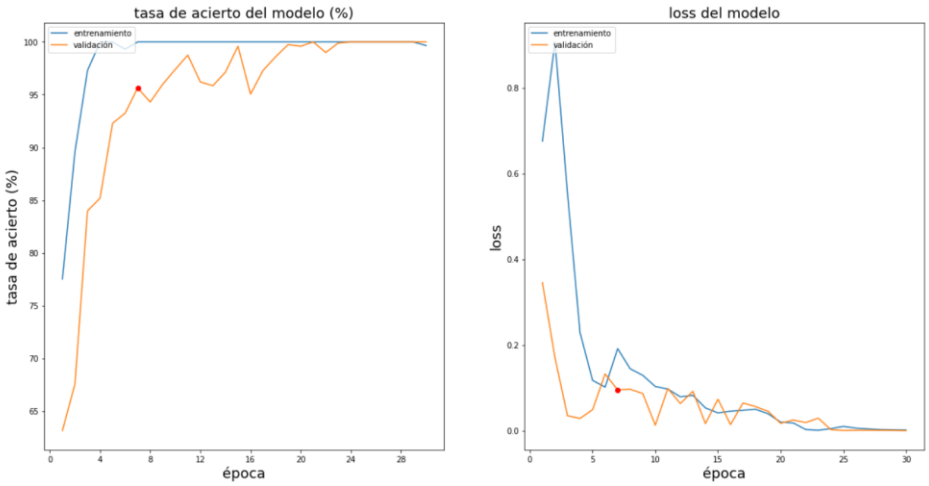

Figura 4-7: *Inception-ResNet* en Hockey

De acuerdo con los resultados se puede concluir que *Inceptionv3* suele ser una peor solución, mientras que *ResNet151v2* e *Inception-ResNet* mejoran ligeramente los resultados, pero aportan gran complejidad al modelo, que se traduce en tiempo de cómputo muy elevado. *ResNet50v2* se establece como la solución más comprometida (balanceada) entre la tasa de resultados y el tiempo de computación.

4.3 Resultados cuantitativos en los cuatro Datasets

4.3.1 Análisis de la calidad de los modelos

El modelo desarrollado se puede evaluar en función del conjunto de datos analizado utilizando las métrica de la mencionada, la Matriz de Confusión. Se ha variado el número de secuencias a utilizar (Tabla 6). El valor porcentual se ha seleccionado en función del tamaño del *Dataset* en cuestión. Quedando: 20% para *HockeyFight* y *RWF-2000* y 25% para *MoviesFight* y *ViolentFlows.* Además, dicha cantidad debe estar balanceada entre clases (sí/ no).

Tabla 6. Número de secuencias empleado de acuerdo con el conjunto de datos

Dataset	**Número total de vídeos**	**Si**	**No**	**Utilizados para test**
HockeyFight	1000	500	500	200
MoviesFight	200	100	100	50
ViolentFlows	400	200	200	100
RWF-2000	2000	1000	1000	400

A continuación, se presentan los informes de clasificación del modelo, usando los datos de la fase de prueba (Tabla 7). Se puede apreciar que en *HockeyFight* se han conseguido buenos resultados, aunque se observa un ligero sesgo hacia violencia (un sesgo conservador). En *MoviesFight* salta a la vista que los resultados son perfectos, esto se debe a que, al ser un conjunto de datos extraído de películas, la violencia está actuada y exagerada, luego el modelo entiende de manera clara y muy sencilla las características a distinguir. *ViolentFlows* por su parte es un *Dataset* complicado, debido a la gran cantidad de individuos presentes en las imágenes, se corrobora lo que era esperado un empeoramiento de los resultados. Por último, aunque *RWF-2000* ofrece vídeos simples (sin aumentos de imagen o movimiento), la larga duración de las escenas puede provocar que no haya violencia en los primeros fotogramas del vídeo y se puede clasificar erróneamente.

Tabla 7. Informe de clasificación del modelo de acuerdo con el conjunto de datos

Dataset		*Precision*	*Recall*	*F_1-score*
HockeyFight	NoViolencia	0,95	0,97	0,95
	Violencia	0,98	0,95	0,96
MoviesFight	NoViolencia	1,00	1,00	1,00
	Violencia	1,00	1,00	1,00
ViolentFlows	NoViolencia	0,96	0,92	0,93
	Violencia	0,92	0,95	0,93
RWF-2000	NoViolencia	0,87	0,92	0,84
	Violencia	0,93	0,87	0,85

4.3.2 Tasas de aciertos del modelo

Para culminar el análisis desde el punto de vista cualitativo, el modelo se puede evaluar en función del conjunto de datos analizado utilizando las métricas de tasas de acierto respecto al conjunto de entrenamiento y de validación (Tabla 8).

Tabla 8. Tasas de acierto final, con respecto al conjunto de entrenamiento y validación

Dataset	Entrenamiento	Validación	Prueba
HockeyFight	100%	98%	96%
MoviesFight	100%	100%	100%
ViolentFlows	100%	96%	94%
RWF-2000	100%	93%	90%

HockeyFight: se observa un aprendizaje progresivo, aunque se encalla la tasa de acierto del conjunto de validación. Sin embargo, el modelo a base de las pérdidas sigue aprendiendo y se observan buenos resultados a partir del *epoch* 15 (Figura 4-8).

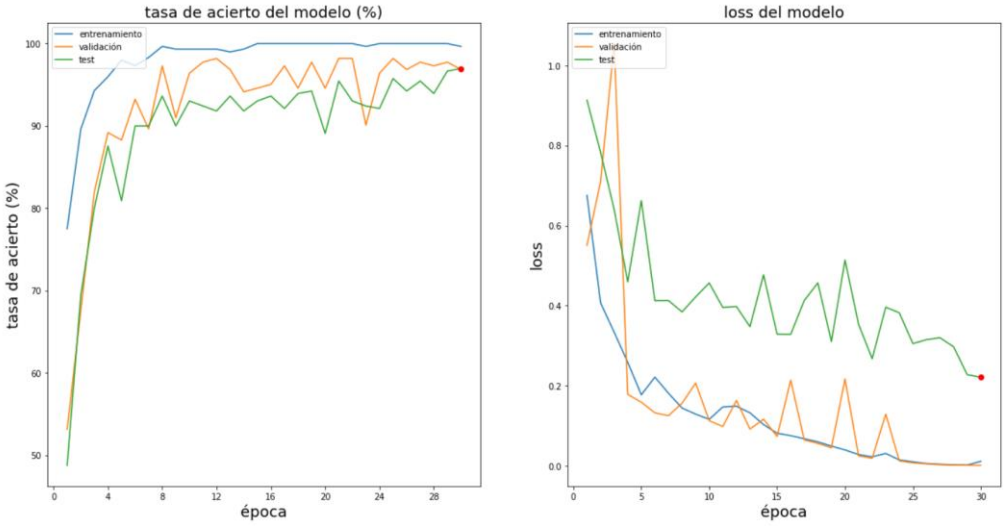

Figura 4-8: Tasa de acierto y pérdidas del modelo sobre *HockeyFight*

MoviesFight: se observa que después de pocos *epochs* se termina comprendiendo el objetivo del entrenamiento y llega a conseguir (en el *epoch* 13) un 100% de acierto en *test* (Figura 4-9). En este *Dataset*, debido a que está basado en películas, se encuentran pocas situaciones donde no queda claro, si se trata de una escena de violencia o no violencia. Por lo tanto, es comprensible la elevada tasa de acierto.

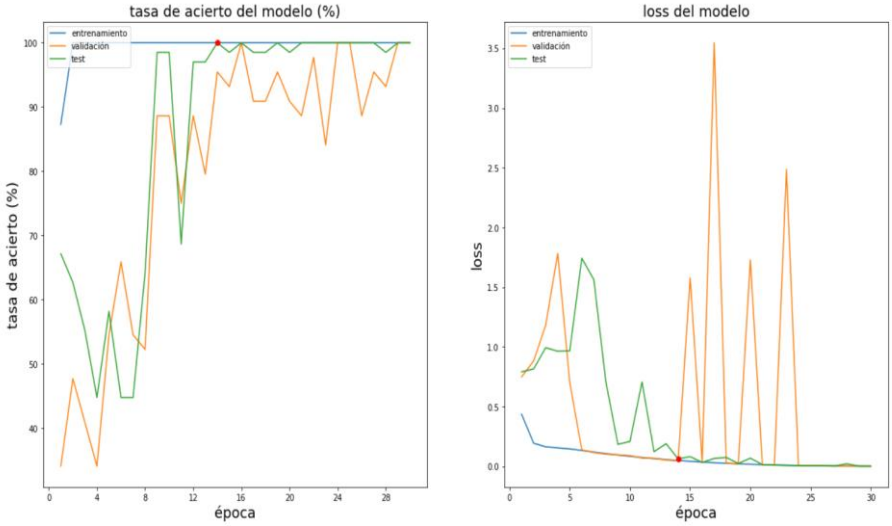

Figura 4-9: Tasa de acierto y pérdidas del modelo sobre *MoviesFight*

ViolentFlows: se consigue una tasa constante de mejora en validación y *test*, yendo más lento a partir del *epoch* 20, debido a que se ha cambiado el *learning rate* a la mitad, permitendo una progresión más lenta para evitar rebotes. Debido a la gran cantidad de personas que hay en estas imágenes, es razonable que le cueste identificar, en qué personas debe fijarse, para establecer el baremo de violencia o no violencia (Figura 4-10).

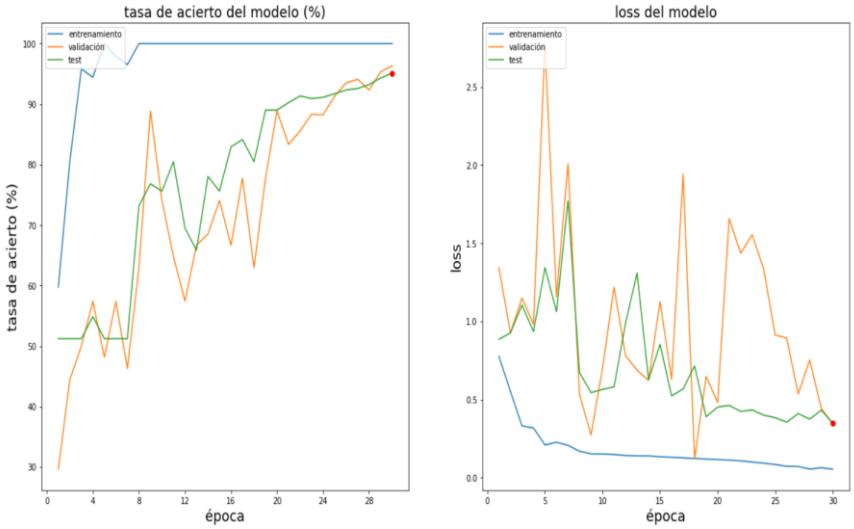

Figura 4-10: Tasa de acierto y pérdidas del modelo sobre *ViolentFlows*

Finalmente, en ***RWF-2000*** se obtiene un resultado similar a *ViolentFlows*, donde se encuentra de nuevo una tasa de aprendizaje reducida sobre el *epoch* 20. Uno de los motivos por los que se debe estos rebotes de aprendizaje es la elevada duración de los vídeos de *RWF-2000* comparada con el resto de *Datasets*, que hace que la violencia no comience hasta varios fotogramas mas tarde, en comparación con el resto de fotogramas donde ocurren prácticamente nada más comenzar el vídeo (Figura 4-11).

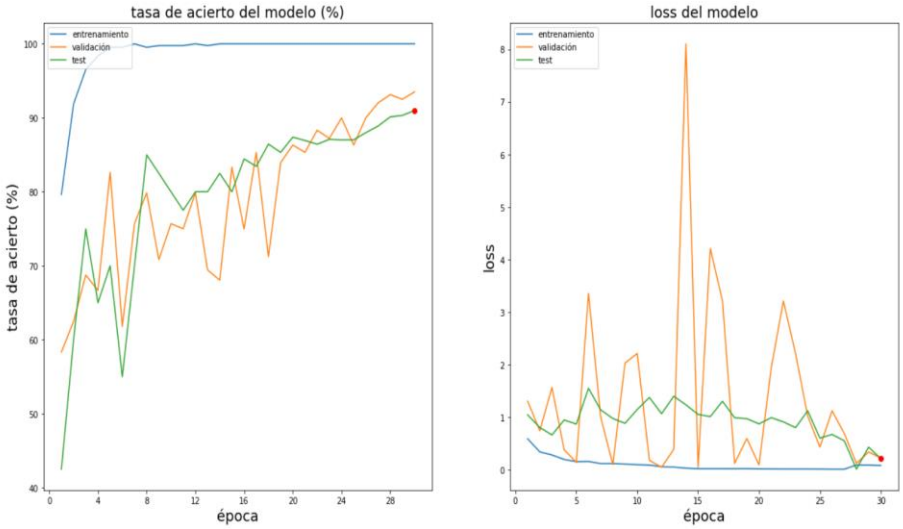

Figura 4-11: Tasa de acierto y pérdidas del modelo sobre *RWF-2000*

4.4 Análisis de los errores del modelo en los cuatro Datasets

En este apartado se resumen los resultados conseguidos con el modelo, se presentan algunos de los fallos detectados, discutiendo su causa y posibles soluciones. En general, se ha visto un aprendizaje satisfactorio en todos los modelos, con un nivel de efectividad menor en el *Dataset RWF-2000*. Viendo ejemplos de secuencias de vídeo, se puede apreciar que se han identificado correctamente numerosas situaciones como violentas, según el contexto. Esto es importante para casos como el *Dataset* de *HockeyFight*, debido a que -como se ha explicado- la acción de deporte, se puede confundir de manera cualitativa como violencia, y es sutil pero importante la diferencia entre estas.

Obsérvese un caso crítico: el impacto entre 2 jugadores en el *Dataset HockeyFights* no242_xvid.avi, que no corresponde a violencia, pero sin contexto, sí se pudiera interpretar como tal. En la Figura 4-12 se puede apreciar cómo hay un choque entre dos jugadores en el centro del fotograma 1, en el fotograma 5 uno de los 2 jugadores empieza a rodar por la pista y este se estabiliza en el fotograma 15. Aún con esta cantidad de movimiento, ha identificado correctamente como “no violencia”.

Fotograma 1 Fotograma 5 Fotograma 15

Figura 4-12: Secuencia de vídeo con no violencia *Dataset HockeyFights*

Otro ejemplo se analiza en el vídeo no77_xvid.avi del *Dataset HockeyFights* donde se puede observar la celebración de un jugador por marcar un gol en el partido (Figura 4-13). Esto implica acercamiento de imagen unido a un movimiento de cámara. El modelo hace una mala predicción (90% posible violencia). En este caso, la secuencia describe claramente el efecto *Ken Burns* [110] desarrollado por el cineasta de igual nombre, para

dar vida a las fotografías, acercándolas lentamente y dándole un movimiento lento entre un punto. Esta mezcla de desplazamiento de cámara (*pan*), junto con un acercamiento (*zoom*) en la imagen, en algunas ocasiones provoca que el modelo se equivoque.

Fotograma 1 Fotograma 5 Fotograma 15

Figura 4-13: Secuencia de vídeo no violento interpretado de forma incorrecta

En el caso de *RWF-2000*, la duración de las secuencias es mucho mayor que el resto de *Datasets* (5 s contra 1 s), por lo que implica una mayor carga de computación. Debido a que el entrenamiento con secuencias de larga duración es inviable (con secuencias de 20 fotogramas se tienen aproximadamente 45 Millones de parámetros a entrenar y más de 8 horas de tiempo de entrenamiento para 30 épocas), *RWF-2000* se ha entrenado con la máxima longitud de secuencia que permitían los recursos, 40 fotogramas.

En el caso de la siguiente secuencia TYRFDVD_1113.avi de *RWF-2000*, formada de un forcejeo entre 2 personas, se ha podido identificar claramente (con un 100% de seguridad en la predicción) que estaba ocurriendo violencia (Figura 4-14).

Fotograma 1 Fotograma 5 Fotograma 15

Figura 4-14: Detección de violencia de forma correcta en *RWF-2000*

Sin embargo, en otra secuencia nuf-d5GugL0_7.avi también de *RWF-2000* (Figura 4-15) donde la acción se está produciendo en un segundo plano, se puede distinguir que está ocurriendo una pelea en la esquina superior izquierda de la imagen, pero está prácticamente omitida. Luego el modelo no ha sido capaz de identificar violencia en la imagen (4% de posibilidades de violencia).

Fotograma 1 Fotograma 5 Fotograma 15

Figura 4-15: Detección de no violencia de forma incorrecta en *RWF-2000*

4.5 Análisis de mapas de calor

En este epígrafe se evalúa, a través de ejemplos, la efectividad de modelo mediante el empleo de un algoritmo de mapas de calor, con la finalidad de validar los ejemplos mostrados en el epígrafe anterior.

Como ejemplo, en la siguiente secuencia, FIGHT_0Ow4cotKOuw_2.avi de *RWF-2000* identificada como violenta, se puede observar que el factor que más ha influenciado a la hora de decidir si la imagen ha sido violenta, es el movimiento de la víctima tras recibir un empujón en el centro de la trifulca (Figura 4-16).

Fotograma 1 Fotograma 5 Fotograma 15

Figura 4-16: Secuencia de vídeo + mapa de calor

Otro ejemplo que demuestra la utilidad de los mapas de calor se puede apreciar en la secuencia previamente analizada FIGHT_nuf-d5GugL0_7.avi del *Dataset RWF-2000* (Figura 4-17). Como se puede se ha mencionado, se produce una pelea entre varias personas en un espacio cerrado. Un hecho característico de esta secuencia es que gran parte de los actos de violencia se están desarrollando fuera del espacio de la imagen. Por lo que, el cerebro humano puede asumir que hay violencia, a partir del análisis del comportamiento de los participantes en el vídeo. La información que proporcionan los mapas de calor no es del todo veraz, el modelo no advierte reacción alguna en los participantes del vídeo, si no que fija su atención en el ordenador. No obstante, se puede apreciar que en algún fotograma -por ejemplo, en el número 15-, que el sistema sí ha dado importancia a los participantes de la acción violenta. Aun así, este instante, no ha tenido suficiente peso en la valoración final como para considerarlo violento (fotogramas 1 y 5).

Fotograma 1 Fotograma 5 Fotograma 15

Figura 4-17: Detección de violencia de forma incorrecta en *RWF-2000*

Como último ejemplo, se reanaliza la secuencia TYRFDVD_1113.avi de *RWF-2000*, surgida de un forcejeo entre 2 individuos (Figura 4-18). El análisis de mapas de calor confirma su interés por los instantes de la secuencia donde se produce un intercambio violento entre las 2 personas, confirmando de esta manera el nivel de conocimiento adquirido por el modelo.

Fotograma 1 Fotograma 5 Fotograma 15

Figura 4-18: Detección de violencia de forma correcta en *RWF-2000*

De acuerdo con los resultados, se puede afirmar que la implementación de mapas de calor es una herramienta eficiente que permite comprobar el funcionamiento adecuado del modelo desarrollado, reafirmado las hipótesis planteadas sobre el presunto conocimiento adquirido. Un problema con esta implementación surge al no tomar en consideración la capa *LSTM*, ya que esta entremezcla información temporal, y no posibilita obtener una correcta representación del tiempo. No se advierte en qué momento del proceso, el modelo se fija en cada detalle de la imagen. La implementación de la librería *GradCAM* no toma en consideración la capa *LSTM*, por lo que los *heatmaps* no reflejan todos los aspectos en los que se está fijando la red para clasificar la imagen.

4.6 Conclusiones relacionadas con la fase experimental

En este apartado se ha comprobado que el empleo de *ResNet50v2* combinada con una *LSTM* permite la creación de un modelo suficientemente robusto, que permite clasificar de manera eficiente actos de violencia, tanto en casos ideales como casos reales. Se han analizado secuencias cuya categorización es errónea, y se han atribuido estos fallos a la aparición de violencia implícita en el vídeo, deducida por el cerebro humano analizando la trama, o a movimientos de cámara unido a gestos, que pueden también parecer violentos sin un contexto que los justifique. Además, el análisis con mapas de calor ha permitido corroborar las características del aprendizaje interno, asegurando un correcto proceso de aprendizaje. La utilización de distintos *Datasets* y la prueba con diferentes modelos ha permitido crear un sistema racional balanceado, computacionalmente sensato y eficaz.

5 Conclusiones y trabajo futuro

5.1 Conclusiones

Para concluir este trabajo, se revisa el trabajo desarrollado en este TFG relacionado con los objetivos planteados al inicio. La consecución de esta meta por pasos está basada en la finalización de un conjunto de tareas:

- Se ha realizado un estudio y análisis de la situación de partida (estado del arte) en relación con la problemática de análisis de imagen. Concretamente en el de detección de violencia, identificando la posibilidad del empleo de redes neuronales profundas como motor principal de un sistema automático de reconocimiento, como solución alternativa a las soluciones planteadas durante el Grado, en el caso de reconocimiento de imágenes.
- Se ha profundizado en la comprensión de las redes neuronales, algunos conceptos previos, sus diferentes componentes, características y tipos; identificando los elementos necesarios para realizar una implementación práctica de estos algoritmos.
- Se ha conseguido una comprensión detallada del funcionamiento de las redes convolucionales y redes recurrentes, aprovechando su capacidad de extraer información visual de manera muy eficiente.
- Se han añadido al modelo técnicas de mejora de aprendizaje, mediante la modificación de los datos de entrenamiento (*data augmentation*) o alteraciones en las conexiones de las capas con el fin de evitar una adquisición de conocimiento no deseado en el problema a resolver.
- Se ha utilizado de *Keras* y *TensorFlow* como una estructura óptima para la aplicación de redes profundas en una implementación en *Python* mediante su uso en *Google Colaboratory*.
- Se han desarrollado e implementado con éxito dos módulos (entrenamiento y análisis) que llevan a la práctica los elementos teóricos analizados.
- Se ha llevado a cabo un proceso comparativo, que posibilita llegar a la conclusión que *ResNet50v2* debe ser utilizado como modelo base del sistema ya que con ello se consigue balance entre eficacia y tiempo de cómputo, en comparación con otras redes soportadas nativamente por *Keras.*
- Se han explicado los resultados del modelo, analizando su comportamiento en diferentes situaciones Viendo que realiza una correcta distinción entre actos violentos y no violentos basándose en términos subjetivos, sin realizar una fijación del movimiento total de los sujetos de la secuencia.
- Se ha demostrado de forma empírica de que el uso elevado de aumento de la imagen (*zoom*) y movimiento de cámara (*pan*) en una secuencia provoca un incremento de errores en la predicción. Aun así, el modelo desarrollado en este trabajo es capaz de diferenciar claramente un cambio de cámara frente a un movimiento de un sujeto en la imagen.
- Se han evaluado las limitaciones que el modelo ha presentado cuando se analizan secuencias cuya violencia no aparece de manera explícita en los fotogramas -el acto violento se encuentra oculto de la vista del espectador-.
- Se ha realizado una implementación de mapas de calor para comprobar la correcta actuación del modelo desarrollado, que ha permitido de forma muy eficiente reafirmar las hipótesis planteadas sobre el presunto conocimiento adquirido del modelo.

Según la investigación hecha en el estado del arte, se ha definido el uso de redes convolucionales como mejor extractor de características en el caso de detección de acción en imagen. La opción que proporciona un mejor balance entre rendimiento y carga computacional es el uso de una red convolucional junto a una red *LSTM*. ResNet ha mostrado ser una arquitectura idónea para solucionar el problema, ofreciendo unos buenos resultados, mientras que *InceptionV3* o *ResNet-Inception* ofrecieron resultados no convincentes. *InceptionV3* es un modelo más simple, por ende, requiere menor capacidad de cómputo, sin embargo, los resultados no son adecuados. Por su parte *ResNet-Inception* es más precisa, pero exige una gran carga de procesamiento. ResNet ha aportado lo mejor de las 2 arquitecturas anteriores para conseguir esta solución.

Según las pruebas experimentales, ha sido beneficioso utilizar aumento de datos y *dropou*t en pequeña medida debido al sobreajuste producido al entrenar. El reentrenamiento de ResNet ha permitido mejorar el rendimiento del proceso, pero ha sido necesario añadir un *kernel regularizer* para evitar perder conocimiento previamente adquirido. Un *kernel* de 256 filtros ha dado los mejores resultados para vídeo a baja resolución.

La novedad de este trabajo consiste en la comparación de esta arquitectura con diversos *Datasets* y el análisis cualitativo y cuantitativo de su rendimiento, soportado por una implementación para vídeo de mapas de calor.

Se han logrado unas tasas de acierto de más del 90% en la totalidad de los casos, lo cual permite corroborar que el modelo es efectivo.

Por lo tanto, los objetivos trazados en el TFG están debidamente resueltos quedando solamente proponer ideas sobre las cuales se podría incidir, para dar continuidad a la investigación.

En comparación con otros trabajos vistos en el estado del arte, este proyecto aporta un análisis de resultados exhaustivo en los *Datasets* propuestos, junto a una implementación de mapas de calor para modelos *ConvLSTM*.

5.2 Trabajo futuro

Esta aproximación a la utilización de redes neuronales como método de reconocimiento de imágenes ha sido satisfactoria, no obstante, es necesario trabajar en la mejora de la capacidad de cómputo de este tipo de sistemas. Un número pequeño de información, como la presente en imágenes de poco tamaño, provoca una gran carga de trabajo. Con una optimización del rendimiento del modelo, se podrían conseguir soluciones cuya resolución de imagen sea mayor, lo cual permitiría crear estructuras de mayor precisión (detalle) y por lo tanto conseguir mayor tasa de acierto en los casos críticos.

Mientras que las redes convolucionales combinadas con *LSTM* han demostrado ser eficientes en la extracción de conocimiento, se deben plantear estrategias alternativas a la hora de analizar secuencias de vídeo, ya que presentan una gran correlación espaciotemporal. Es posible una aproximación a la solución del problema mediante el estudio y aplicación de los algoritmos de codificación de vídeo como *MPEG-4* (dominios transformados), que reducen de manera drástica la información transmitida sin provocar una pérdida significativa de información cualitativa.

Se debe reconsiderar el empleo de la diferencia de fotogramas como entrada del modelo, una vez demostrado el conocimiento adquirido en los fotogramas de manera individual.

Referencias

Bibliografía

[1] Zhang,W., Xue, X., Sun,Z., Guo, Y., ChiM., Lu, H. (2007) *Efficient Feature Extraction for Image Classification.* IEEE 11th International Conference on Computer Vision, Rio de Janeiro, pp. 1-8, doi: 10.1109/ICCV.2007.4409058.
https://ieeexplore.ieee.org/document/4409058

[2] Wikipedia (2020) *Haar-like feature*. Last edited on 06 May 2020
https://en.wikipedia.org/wiki/Haar-like_feature
Último acceso: 2020/06/10

[3] Saha, S., Singh, G., Sapienza, M., Torr, P., Cuzzolin, F. (2016). *Deep Learning for Detecting Multiple Space-Time Action Tubes in Videos*. 58.1-58.13. 10.5244/C.30.58.
https://arxiv.org/pdf/1608.01529.pdf

[4] Dalal, N., Triggs, B., (2005). *Histograms of Oriented Gradients for Human Detection*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005).
https://www.researchgate.net/publication/281327886_Histograms_of_Oriented_Gradients_for_Human_Detection/citation/download

[5] Chaudhry, R., Ravichandran, A., Hager, G., Vidal, R., (2009). *Histograms of oriented optical flow and Binet-Cauchy kernels on nonlinear dynamical systems for the recognition of human actions.* 1932-1939.
https://www.researchgate.net/publication/221362187_Histograms_of_oriented_optical_flow_and_Binet-Cauchy_kernels_on_nonlinear_dynamical_systems_for_the_recognition_of_human_actions

[6] Pietikäinen, M. (2005) *Image Analysis with Local Binary Patterns*
https://www.researchgate.net/publication/220809552_Image_Analysis_with_Local_Binary_Patterns

[7] Peng, X., Qiao, Y., Peng. Q,. Xianbiao Q (2013). *Exploring Motion Boundary based Sampling and Spatial-Temporal Context Descriptors for Action Recognition*
http://www.bmva.org/bmvc/2013/Papers/paper0059/paper0059.pdf

[8] Poppe, R. (2010) *A survey on vision-based human action recognition*. Image and Vision Computing 28, 976–990
https://www.sciencedirect.com/science/article/abs/pii/S0262885609002704

[9] Giannakopoulos, T., Pikrakis, A., & Theodoridis, S. (2010). *A multimodal approach to violence detection in video sharing sites.* In Pattern Recognition (ICPR), 2010 20th International Conference on (pp. 3244-3247). IEEE.
https://www.researchgate.net/publication/220931681_A_Multimodal_Approach_to_Violence_Detection_in_Video_Sharing_Sites

[10] Nievas, E. B., Suarez, O. D., García, G. B., Sukthankar, R. (2011). *Violence detection in video using computer vision techniques.* In Computer Analysis of Images and Patterns (pp. 332-339). Springer Berlin Heidelberg.
https://link.springer.com/chapter/10.1007/978-3-642-23678-5_39

[11] Ming-yu, C., Hauptmann., A. (2018) *Mosift: Recognizing human actions in surveillance videos*. (2009).
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.962.7080&rep=rep1&type=pdf

[12] Eyben, F., Weninger, F., Lehment, N., Schuller, B., Rigoll, G. (2013*). Affective video retrieval: Violence detection in Hollywood movies by large-scale segmental feature extraction.* PloS one, 8(12), e78506.
https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0078506

[13] Mironică, I., Duta, I, Ionescu, B., Sebe, N. (2015). *A modified vector of locally aggregated descriptors approach for fast video classification.* Multimedia Tools and Applications. 75. 10.1007/s11042-015-2819-7
https://link.springer.com/article/10.1007/s11042-015-2819-7

[14] Dehe, Y., Jingguo, L., Danlu, Z., Jingfa, Z., Jing., Y. (2019) *Extracting multi-features and optimizing feature space with sparse auto-encoder over WorldView-2 images*, International Journal of Remote Sensing, 40:16, 6418-6443, DOI: 10.1080/01431161.2019.1594431
https://www.tandfonline.com/doi/abs/10.1080/01431161.2019.1594431?journalCode=tres20&

[15] Voulodimos, A., Doulamis, N., Doulamis, A., Protopapadakis, E. (2018) *Deep Learning for Computer Vision: A Brief Review*
https://www.hindawi.com/journals/cin/2018/7068349/

[16] Rumelhart, D. E., Hinton, G. E., y Williams, R. J. (1986) *Learning representations by back-propagating errors*. Nature 323, 533–536.
https://www.nature.com/articles/323533a0
[17] Kostadinow, S. (2019) *Understanding Backpropagation Algorithm*
https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd
[18] LeCun, Y., Boser, B., Denker,J., Henderson, D. Howard, R., Hubbard,W., Jackel, L. (1989) *Backpropagation Applied to Handwritten Zip Code Recognition*. Neural Computation, vol. 1, no. 4, pp. 541-551, Dec. 1989, doi: 10.1162/neco.4.541.
https://ieeexplore.ieee.org/document/6795724
[19] LeCun, Y., Bottoy, L., Bengio, Y., Haffner. P., (1998) *Gradient-based learning Applied to Document Recognition.* IEEE International Conference
http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
[20] Fang, S. (2020) *Review of LeNet-5: How to design the architecture of CNN.* Towards Data Science.
https://towardsdatascience.com/review-of-lenet-5-how-to-design-the-architecture-of-cnn-8ee92ff760ac
Último acceso: 2020/06/10
[21] Poznanski, A., Wolf, L.(2016) *CNN-N-Gram for Handwriting Word Recognition*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
https://ieeexplore.ieee.org/document/7780622
[22] Olah, C. (2015) *Understanding LSTM Networks*
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Último acceso: 2020/06/10
[23] Hochreiter, S., Schmidhuber, J. (1997). *Long Short-term Memory*. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.
https://www.researchgate.net/publication/13853244_Long_Short-term_Memory
[24] Misra, A. (2019) *Using RNNs for Machine Translation*. Towards data science.
https://towardsdatascience.com/using-rnns-for-machine-translation-11ddded78ddf
Último acceso: 2020/06/10
[25] Sotner, D. Müller, L. (2013) *Application of LSTM Neural Networks in Language Modelling*. International Conference on Text, Speech and Dialogue
https://link.springer.com/chapter/10.1007/978-3-642-40585-3_14
[26] Issa, A. (2019) *Generating Original Classical Music with an LSTM Neural Network and Attention*. Medium
https://medium.com/@alexissa122/generating-original-classical-music-with-an-lstm-neural-network-and-attention-abf03f9ddcb4
[27] Nayak, M. (2018) *An Intuitive Introduction of Boltzmann Machine*
https://medium.com/datadriveninvestor/an-intuitive-introduction-of-boltzmann-machine-8ec54980d789
Último acceso: 2020/06/10
[28] Duran, J. (2019) *Técnicas de Regularización Básicas para Redes Neuronales.* MetaDatos
https://medium.com/metadatos/técnicas-de-regularización-básicas-para-redes-neuronales-b48f396924d4
[29] Fischer, A., Igel, C. (2012) *An Introduction to Restricted Boltzmann Machines*. Iberoamerican Congress on Pattern Recognition
https://link.springer.com/chapter/10.1007/978-3-642-33275-3_2
[30] Hinton, G., Osindero, S., The, Y., (2006) *A fast learning algorithm for deep belief nets.* Neutral Computation Vol 18, 7
https://dl.acm.org/doi/10.1162/neco.2006.18.7.1527
[31] Salakhutdinov, R.. Hinton, G. (2009) *Deep Boltzmann Machines.* Department of Computer Science. University of Toronto
http://www.cs.toronto.edu/~fritz/absps/dbm.pdf
[32] Bladi, P. (2012) *Autoencoders, Unsupervised Learning, and Deep Architectures.* JMLR: Workshop and Conference Proceedings 27:37–50
http://proceedings.mlr.press/v27/baldi12a/baldi12a.pdf
[33] Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P. (2008) *Extracting and Composing Robust Features with Denoising Autoencoders.* CS. Toronto.
https://www.cs.toronto.edu/~larocheh/publications/icml-2008-denoising-autoencoders.pdf
[34] Krizhevsky, A. Sutskeve, I. y Hinton, G. (2012) *ImageNet Classification with Deep Convolutional Neural Networks.*

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[35] Llamas, J., Lerones, P., Medina, R., Zalama, E., y Gómez, J. (2017). *Classification of Architectural Heritage Images Using Deep Learning Techniques*. Applied Sciences. 7. 992. 10.3390/app7100992. https://www.mdpi.com/2076-3417/7/10/992/htm

[36] Chellapilla, K., Puri, S., Simard, P. (2006) *High Performance Convolutional Neural Networks for Document Processing*. Tenth International Workshop on Frontiers in Handwriting Recognition, Université de Rennes 1, Oct 2006, La Baule (France). ffinria-00112631f https://hal.inria.fr/file/index/docid/112631/filename/p1038112283956.pdf

[37] Cires, D., Meier, D. Masci, J., Gambardella, L., Schmidhuber. J.(2011) *Flexible, High Performance Convolutional.* Neural Networks for Image Classification. IDSIA, USI and SUPSI http://people.idsia.ch/~juergen/ijcai2011.pdf

[38] Fukushima, K., (1980) *Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position.* Biol. Cybernetics 36, 193 202 https://www.rctn.org/bruno/public/papers/Fukushima1980.pdf

[39] Weng, J., Ahuja, N., Huang, T (1993). *Learning recognition and segmentation of 3D objects from 2D images*. Proc. 4th International Conf. Computer Vision: 121–128. https://www.cse.msu.edu/~weng/research/CresceptronICCV1993.pdf

[40] He, K, Zhang, X., Ren S. and Sun J.(2015) *Deep Residual Learning for Image Recognition.* Cornell University. NY. https://arxiv.org/abs/1512.03385

[41] Wikipedia (2020) *Machine learning*. Last edited on 29 May 2020 https://en.wikipedia.org/wiki/Machine_learning Último acceso: 2020/06/10

[42] Mitchell, T. (1997) *Machine Learning.* McGraw Hill http://www.cs.cmu.edu/~tom/mlbook.html

[43] Sunil R. (2017) *Commonly used Machine Learning Algorithms (with Python and R Codes*). Analytics Vidhya. https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/ Último acceso: 2020/06/10

[44] Gonzalez, L. (2019) *Árboles de Decisión Clasificación – Teoría* https://ligdigonzalez.com/arboles-de-decision-clasificacion-teoria-machine-learning/ Último acceso: 2020/06/10

[45] Swaminathan, S. (2018) *Logistic Regression — Detailed Overview*. Towards Data Science https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc Último acceso: 2020/06/10

[46] Patel, S. (2017) *Chapter 2: SVM (Support Vector Machine) — Theory*. Medium https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72 Último acceso: 2020/06/10

[47] Brownlee, J. (2019) *A Gentle Introduction to Bayesian Belief Networks.* Machine Learning Mastery https://machinelearningmastery.com/introduction-to-bayesian-belief-networks/ Último acceso: 2020/06/10

[48] Chandra, A. (2018) *Perceptron: The Artificial Neuron (An essential upgrade to the McCulloch-Pitts neuron).* Towards Data Science https://towardsdatascience.com/perceptron-the-artificial-neuron-4d8c70d5cc8d Último acceso: 2020/06/10

[49] FD (2017) *Batch normalization in Neural Networks.* Towards Data Science https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c Último acceso: 2020/06/10

[50] Sinnott, R., Sun, Y. (2016) *A Case Study in Big Data Analytics*. Big Data https://www.sciencedirect.com/topics/computer-science/sigmoid-function Último acceso: 2020/06/10

[51] Polamuri, S., (2017) *Difference Between Softmax Function and Sigmoid Function* . Dataaspitant https://dataaspirant.com/2017/03/07/difference-between-softmax-function-and-sigmoid-function/

[52] Brownlee, J. (2019) *A Gentle Introduction to the Rectified Linear Unit (ReLU).* Machine Learning Mastery. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ Último acceso: 2020/06/10

[53] Alberti, M. (2018) *Redes neuronales capsulares*. Deep Learning – ES
https://www.deeplearningitalia.com/redes-neuronales-capsulares/
Último acceso: 2020/06/10
[54] Zhang, L., Hong, L., King, X. (2019) *Evolving feedforward artificial neural networks using a two-stage approach*. Neurocomputing Volume 360, 30 September, Pages 25-36
https://www.sciencedirect.com/science/article/pii/S0925231219309191
[55] Sherstinsky, A. (2020) *Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network*. Cornell University
https://arxiv.org/abs/1808.03314
[56] Larrañaga, P., Inza I. y Moujahid, A. (2020) *Redes Neuronales*. Universidad del País Vasco
http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/
[57] Smart Panel (2020) *¿Qué es el Deep Learning?*
https://www.smartpanel.com/que-es-deep-learning/
Último acceso: 2020/06/10
[58] Pihlajamäki, T. (2020) *Multi-resolution Short-time Fourier Transform Implementation of Directional Audio Coding*
https://www.researchgate.net/figure/The-calculation-of-convolution-Two-signals-red-and-blue-are-convolved-with-each_fig10_267239829
[59] Cowley J. (2018) *Redes neuronales convolucionales. Utilizar Python para implementar una red sencilla que clasifica dígitos escritos a mano*. IBM
https://www.ibm.com/developerworks/ssa/library/cc-convolutional-neural-network-vision-recognition/index.html
[60] Sinhal, K. and Sachan, A. (2020) *Understand Resnet / AlexNet / Vgg / Inception.*
https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/
Último acceso: 2020/06/10
[61] Katarzyna, J., Wojciech, M., (2017) *On Loss Functions for Deep Neural Networks in Classification*. Theoretical Foundations of Machine Learning 2017
https://arxiv.org/pdf/1702.05659.pdf
[62] He, K, Zhang, X., Ren S. and Sun J. (2016) *Identity Mappings in Deep Residual Networks.* Cornell University. NY.
https://arxiv.org/abs/1603.05027
[63] Szegedy, C. (2014) *Building a deeper understanding of images*. Google Al
https://ai.googleblog.com/2014/09/building-deeper-understanding-of-images.html
[64] Cao, X. Wipf, D., Wen,F. Duan,G.and Sun,J. (2013) *A practical transfer learning algorithm for face verification*," in Proceedings of the 14th IEEE International Conference
https://www.microsoft.com/en-us/research/wp-content/uploads/2013/01/TransferLearning.pdf
[65] Lawrence, S. Giles, C. Tsoi, A., and Back, A. (1997) *Face recognition: a convolutional neural-network approach*, IEEE Transactions on Neural Networks and Learning Systems, vol. 8,no. 1, pp. 98–113.
https://dl.acm.org/doi/10.1109/72.554195
[66] Schroff, F., Kalenichenko, D. Philbin, J, (2015) *FaceNet: A Unified Embedding for Face Recognition and Clustering*. Cornell University
https://arxiv.org/abs/1503.03832
[67] Taigman, Y., Yang, M., Ranzato M., Wolf, L. (2014) *DeepFace: Closing the Gap to Human-Level Performance in Face Verification*, IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 1701-1708, doi: 10.1109/CVPR.2014.220.
https://ieeexplore.ieee.org/document/6909616
[68] Gan, C., Wang, N., Yang, Y., Yeung, D., Hauptmann, A., *DevNet: A Deep Event Network for multimedia event detection and evidence recounting 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, 2015, pp. 2568-2577, doi: 10.1109/CVPR.2015.7298872.
https://ieeexplore.ieee.org/document/7298872
[69] Kautz, T., Groh, B., Hannink, J. Jensen, U., Strubberg, H., Eskofier, B. (2017) *Activity recognition in beach volleyball using a DEEp Convolutional Neural NETwork: leveraging the potential of DEEp Learning in sports*," Data Mining and Knowledge. Discovery, vol. 31, no. 6, pp. 1678–1705.
https://dl.acm.org/doi/10.1007/s10618-017-0495-0
[70] Ronao, C., Cho,S. (2016) *Human activity recognition with smartphone sensors using deep learning neural networks*. Expert Systems with Applications, vol. 59, pp. 235–244
https://www.sciencedirect.com/science/article/abs/pii/S0957417416302056

[71] Kitsikidis, A., Dimitropoulos, K., Douka, S., Grammalidis, N. (2014). *Dance analysis using multiple Kinect sensors*. VISAPP- Proceedings of the 9th International Conference on Computer Vision Theory and Applications. 2. 789-795.
https://www.researchgate.net/publication/287882481_Dance_analysis_using_multiple_kinect_sensors/citation/download

[72] Felzenszwalb, P.F., Huttenlocher, D.P. (2005). *Pictorial Structures for Object Recognition*. International Journal of Computer Vision 61, 55–79
https://doi.org/10.1023/B:VISI.0000042934.15159.49

[73] LISA Lab (2018) *Convolutional Neural Networks (LeNet).* DeepLearning 0.1 docs.
http://deeplearning.net/tutorial/lenet.html
Último acceso: 2020/06/10

[74] Karpathy, A., Toderici, G., Shetty, S., Leung,T., Sukthankar, R. Fei-Fei, L. (2014) *Large-scale Video Classification with Convolutional Neural Networ*ks. Proceedings of International Computer Vision and Pattern Recognition, IEEE
https://research.google/pubs/pub42455/

[75] Simonyan, K., Zisserman, A., (2014) *Two-Stream Convolutional Networks for Action Recognition in Videos.* Cornell University
https://arxiv.org/abs/1406.2199

[76] Eskinder, A., Sanjay., D. (2016). *Motion Estimation in Video Coding using Simplified Optical Flow Technique*. International Arab Journal of Information Technology. 13. 770-776.
https://www.researchgate.net/publication/316345345_Motion_Estimation_in_Video_Coding_using_Simplified_Optical_Flow_Technique/citation/download

[77] Warren, W. (2010). *Optic Flow*. 10.1016/B978-012370880-9.00311-X.
https://www.researchgate.net/publication/286038595_Optic_Flow/citation/download

[78] Tran, D., Bourdev, L., Fergus, R., Torresani, L., Paluri, M. (2015) L*earning Spatiotemporal Features with 3D Convolutional Networks*
https://arxiv.org/pdf/1412.0767.pdf

[79] Sudhakaran, S., Lanz, O., (2017) *Learning to Detect Violent Videos using Convolutional Long Short-Term Memory*
https://arxiv.org/pdf/1709.06531.pdf

[80] Ashikin, A., Norhalina., S (2016). *A Review on Violence Video Classification Using Convolutional Neural Networks*. 10.1007/978-3-319-51281-5_14.
https://www.researchgate.net/publication/312031821_A_Review_on_Violence_Video_Classification_Using_Convolutional_Neural_Networks

[81] Fath U Min Ullah, F., Ullah, A. Muhammad, K., Ul, I., Baik, S. (2019) *Violence Detection Using Spatiotemporal Features with 3D Convolutional Neural Network*
https://doi.org/10.3390/s19112472

[82] M. Ramzan, M. (2019) *A Review on State-of-the-Art Violence Detection Techniques*. IEEE Access, vol. 7, pp. 107560-107575, 2019, doi: 10.1109/ACCESS.2019.2932114.
https://ieeexplore.ieee.org/abstract/document/8782115

[83] Li. C., Zhu, L., Zhu, D., Chen, J., Pan, Z., Li, X., Wang, B. (2018) *End-to-end Multiplayer Violence Detection based on Deep 3D CNN*
https://dl.acm.org/doi/10.1145/3301326.3301367

[84] Zhou, B., Khosla, A., Lapedriza, A., Oliva, A, Torralba, A. (2016) *Learning Deep Features for Discriminative Localization*. IEEE
http://cnnlocalization.csail.mit.edu/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf

[85] Huang, G., Liu, Z., Van der Maatern, L., Weinberger, Q., (2016) *Densely Connected Convolutional Networks*. Cornell University
https://arxiv.org/abs/1608.06993

[86] GitHub (2020) *Deep Residual Learning for Image Recognition*
https://github.com/keras-team/keras-applications/blob/master/keras_applications/resnet50.py
Último acceso: 2020/06/10

[87] *Python Docs: Functional Programming HOWTO*
https://docs.python.org/3/howto/functional.html
Último acceso: 2020/06/10

[88] Shorten, C., Khoshgoftaar, T. (2019) *A survey on Image Data Augmentation for Deep Learning*. Big Data.
https://link.springer.com/content/pdf/10.1186/s40537-019-0197-0.pdf

[89] Srivastava, N., Hinton, G., Krizhevsky, A. Sutskever, I., Sutskever R. (2014) *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Journal of Machine Learning Research 15. 1929-1958

https://pdfs.semanticscholar.org/6c8b/30f63f265c32e26d999aa1fef5286b8308ad.pdf?_ga=2.205586579.1807662726.1591718751-172242360.1591273965
[90] Keras (2020) *ResNet and ResNetV2*
https://keras.io/api/applications/resnet/#resnet-and-resnetv2
Último acceso: 2020/06/10
[91] TensorFlow (2020) *An end-to-end open source machine learning platform*
https://www.tensorflow.org/
Último acceso: 2020/06/10
[92] Wikipedia (2020) *Theano.* Last edited on 14 September 2012
https://en.wikipedia.org/wiki/Theano
Último acceso: 2020/06/10
[93] Wikipedia (2020) *PyTorch*. Last edited on 29 April 2020
https://en.wikipedia.org/wiki/PyTorch
Último acceso: 2020/06/10
[94] Blunsden, S., Fisher, R., (2009) *The behave video dataset: ground truthed video for multi-person behavior classification*
http://homepages.inf.ed.ac.uk/rbf/PAPERS/unfbehavedata.pdf
[95] Rota, P., Conci, N., Sebe, N., Rehg, J. (2015) *Real-Life Violent Social Interaction Detection*
http://mhug.disi.unitn.it/wp-content/uploads/2015/papers/2015/Rota-ICIP15.pdf
[96] Demarty, C., Penet, C., Soleymani, M., G. Gravier, G., (2015) *Vsd, a public dataset for the detection of violent scenes in movies: design, annotation, analysis and evaluation*. Multimedia Tools and Applications, vol. 74, no. 17, pp. 7379–7404, 2015
https://link.springer.com/article/10.1007/s11042-014-1984-4
[97] M. Perez, M., Kot, A., Rocha, A. (2019) *Detection of real-world fights in surveillance videos.* in ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, pp. 2662–2666
https://ieeexplore.ieee.org/document/8683676
[98] Nievas, E., Suarez, O., Garcia, G., Sukthankar, R. (2011) *Hockey fight detection dataset*. Computer Analysis of Images and Patterns. Springer, pp. 332–339
http://visilab.etsii.uclm.es/personas/oscar/FightDetection/
[99] Nievas, E., Suarez, O., Garcia, G., Sukthankar, R. (2011) *Movies fight detection dataset.* Computer Analysis of Images and Patterns. Springer, 2011, pp. 332–339.
http://visilab.etsii.uclm.es/personas/oscar/FightDetection/
[100] Hassner, T., Itcher,Y., Kliper-Gross, O., (2012) *Violent flows: Real-time detection of violent crowd behavior*. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. IEEE, pp. 1–6.
https://ieeexplore.ieee.org/document/6239348
[101] Yun, K., Honorio, J., Chattopadhyay, D., Berg, T., Samaras, D., (2012) *Two-person interaction detection using body-pose features and multiple instance learning*. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. IEEE, 2012, pp. 28–35.
http://people.csail.mit.edu/jhonorio/actrec_hau3d12.pdf
[102] Sultani, W., Chen, C., Shah, M., (2018) *Real-world anomaly detection in surveillance videos*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6479–6488.
https://arxiv.org/abs/1801.04264
[103] Cheng, M., Cai, K., Li, M. (2020) *RWF-2000: An Open Large Scale Video Database for Violence Detection*
https://arxiv.org/pdf/1911.05913.pdf
[104] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z. (2015) *Rethinking the Inception Architecture for Computer Vision.* Cornell University. NY.
https://arxiv.org/abs/1512.00567
[105] Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A. (2016) *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning.* Cornell University. NY.
https://arxiv.org/abs/1602.07261
[106] Krishnan, M. (2018) *Understanding the Classification report through sklearn*
https://muthu.co/understanding-the-classification-report-in-sklearn/
Último acceso: 2020/06/10
[107] Selvaraju, R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D. (2019) *Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization*. Cornell University. NY.
https://arxiv.org/abs/1610.02391
[108] Mishra, D. (2019) *Demystifying Convolutional Neural Networks using GradCam*

https://towardsdatascience.com/demystifying-convolutional-neural-networks-using-gradcam-554a85dd4e48
Último acceso: 2020/06/10
[109] Rosebrock, A. (2020) *Grad-CAM: Visualize class activation maps with Keras, TensorFlow, and Deep Learning.*
https://www.pyimagesearch.com/2020/03/09/grad-cam-visualize-class-activation-maps-with-keras-tensorflow-and-deep-learning/
Último acceso: 2020/06/10
[110] Chinnathambi, K. (2015) *The Ken Burns Effect using CSS Animations*
https://www.kirupa.com/html5/ken_burns_effect_css.htm
Último acceso: 2020/06/10

Licencias de utilización de los Datasets

Se permite el empleo de todos los *Datasets* utilizados con fines académicos, siendo necesario referenciar al autor original del *Dataset* como sigue:

A. ***HockeyFight*** E. B. Nievas, O. D. Suarez, G. B. Garcia, and R. Sukthankar, *"Hockey fight detection dataset,"* in Computer Analysis of Images and Patterns. Springer, 2011, pp. 332–339.

B. ***MoviesFight*** E. B. Nievas, O. D. Suarez, G. B. Garcia, and R. Sukthankar, "*Movies fight detection dataset,"* in Computer Analysis of Images and Patterns. Springer, 2011, pp. 332–339.

C. ***ViolentFlows*** T. Hassner, Y. Itcher, and O. Kliper-Gross, *"Violent flows: Real-time detection of violent crowd behavior,"* in 2012 IEEE Computer Society. Conference on Computer Vision and Pattern Recognition Workshops. IEEE, 2012, pp. 1–6.

D. ***RWF-2000*** Ming Cheng, Kunjing Cai, and Ming Li. *"RWF-2000: An Open Large-Scale Video Database for Violence Detection."* arXiv preprint arXiv:1911.05913 (2019)

Glosario

Índice de siglas

	Inglés	**Español**
ANN	*Artificial Neural Networks*	Redes Neuronales Artificiales (RNA)
API	*Application Programming Interface*	Interfaz de Programación de Aplicaciones
BN	*Batch Normalization*	Normalización por Lotes
CAM	*Class Activation Mapping*	Mapeo de Activación de Clases
CNN	*Convolutional Neural Network*	Red Neuronal Convolucional (RNC)
ConvNet	*Convolutional Neural Network*	Red Neuronal Convolucional (RNC)
Conv2D	*Convolutional Operation in 2D*	Operación Convolucional en 2D
CPU	*Central Process Unit*	Unidad de Procesamiento Central
CUDA	*Compute Unified Device Architecture*	Arquitectura Unificada de Dispositivos de Cómputo
DAG	*Directed Acyclic Graph*	Grafo Acíclico Dirigido
DBN	*Deep Belief Networks*	Redes de creencias profundas
DBM	*Deep Boltzmann Machines*	máquinas de Boltzmann profundas
FANN	*Feed-forward Artificial Neural Networks*	Redes Neuronales de Retroalimentación
FCL	*Fully-Connected Layers*	Capas Completamente Conectadas
VGG	*Face Descriptor*	descriptor de caras
GPU	*Graphics Processing Units*	Unidades de Procesamiento de Gráficos
HOG	*Histogram of Oriented Gradients*	Histogramas de Gradientes Orientados
HOOF	*Histogram of Oriented Optic Flow*	Histogramas de Flujos Ópticos Orientados
IDSIA	*Institute for Artificial Intelligence Research (Istituto Dalle Molle di Studi sull'Intelligenza Artificiale Dalle Molle)*	Instituto Dalle Molle para la Investigación de Inteligencia Artificial (Ticino, Suiza).
k-NN	*k-nearest neighbors*	clasificador de vecinos más próximos
LBP	*Local Binary Patterns*	Patrones binarios locales
LSTM	*Long Short Term Memory*	Memoria a Corto y Largo Plazo
MAE	*Mean Absolute Error*	Error Absoluto Medio
MBH	*Motion Boundary Histogram*	Histogramas de Perímetro de Movimiento
MCP	*McCulloch and Pitts neuron model*	Modelo neuronal McCulloch y Pitts
MPEG	*Moving Picture Experts Group*	formato de vídeo digital estandarizado por un grupo de trabajo de expertos de la Organización Internacional de Normalización (ISO) y la Comisión Electrotécnica Internacional para establecer estándares para el audio y la transmisión vídeo
MSE	*Mean Squared Error*	Error Medio Cuadrático
ReLU	*Rectifier Linear Unit*	Unidad Lineal Rectificadora
RBM	*Restricted Boltzmann Machine*	máquina de Boltzmann restringida
RGB	*Red, Green, Blue*	composición del color en términos de la intensidad de los colores primarios de la luz
RNN	*Recurrent Neural Networks*	Redes Neuronales Recurrentes
SGD	*Stochastic Descent Gradient*	Gradiente de descenso estocástico
SVM	*Support Vector Machines*	Máquinas de Soporte Vectorial
Grad- CAM	*Gradient-weighted Class Activation Mapping*	Asignación de activación de clase ponderada por gradiente

Terminología en inglés

Inglés	**Español**
Adam	algoritmo para la optimización basada en gradiente del objetivo estocástico
automatic hyperparameter tuning	ajuste automático de hiperparámetros
Autoencoders	tipo de red neuronal artificial que se utiliza para aprender codificaciones de datos eficientes de manera no supervisada.
backpropagation errors	retropropagación de errores
Batch Normalization	normalización por lotes
Bias	sesgo algorítmico
Boltzmann Machine	tipo de red neuronal recurrente estocástica
categorical cross entropy	clasificación la entropía cruzada
Classification report	matriz de confusión
convolutional layers	capas convolucionales
data augmentation	datos aumentados
datasets	juegos de datos
deep learning	aprendizaje profundo
DeepFace	sistema de reconocimiento facial de aprendizaje profundo creado por un equipo de desarrollo en Facebook
denoising autoencoder	autoencoder de eliminación de ruido
DenseNets	redes densas
dropout	marginado
early fusion	fusión temprana
end-to-end	de extremo a extremo
epochs	épocas (ciclo complete en que un modelo recorre un juego de datos)
FaceNet	sistema de reconocimiento facial desarrollado en 2015 por investigadores de Google
False Negative	Falso negativo, cuando un caso fue positivo, pero se predijo negativo
False Positive	Falso positivo, cuando un caso fue negativo, pero se predijo positivo
feature engineering	técnicas de ingeniería utilizadas para trabajar sobre los atributos
feature learning	aprendizaje de características o aprendizaje de representación
feed-forward	alimentación positiva
forward propagation	propagación hacia adelante
finetuning	afinación
float32	conocido también como binary32, es uno de los formatos descritos en los estándares técnicos IEEE 754-1985, IEEE 754-2008, IEEE 854-1987 y ISO/IEC/IEEE 60559:2011.
frame	fotograma
framework	entorno de trabajo
generator	generador
Google Colaboratory	entorno de máquinas virtuales de Google
gradient descent	descenso de gradiente
Hebbian learning rule	regla de aprendizaje que especifica cuánto se debe aumentar o disminuir el peso de la conexión entre dos unidades en proporción al producto de su activación.
hidden nodes	nodos ocultos
HighwayNets	red de carreteras
HockeyFight	juego de datos de partidos de Hockey
identity shortcut connections	atajos de identidad
Identity Function	Función identidad (*Null Function*)
ImageNet	base de datos de imágenes organizada según la jerarquía de *WordNet*, en la que cada nodo está representado por cientos y miles de imágenes.
Inception	técnica utilizada originalmente en *LeNet*, para identificar patrones en imágenes. Se utiliza en redes neuronales convolucionales para permitir un cálculo más eficiente y redes más profundas a través de una reducción de dimensionalidad con convoluciones apiladas 1×1.
input	entrada
input nodes	nodos de entrada
Ken Burns effect	técnica de edición de vídeo consistente al hacer zoom y mover una imagen originalmente fija
Keras	biblioteca de Redes Neuronales de código abierto escrita en Python
kernel	método para el análisis de patrones, cuyo miembro más conocido son las Máquinas de Vectores de Soporte
kernel regularizers	aprendizaje inductivo regularizado
label	categoría etiqueta
late fusion	fusión tardía
learning rate	tasa de aprendizaje

Inglés	Español
LeNet	estructura de red neuronal convolucional propuesta por Yann LeCun y colaboradores en 1998. En general, LeNet se refiere a lenet-5 y es una red neuronal convolucional simple.
loss	pérdida
machine learning	aprendizaje automático (RAE)
MaxPooling	capa de agrupación por máximo
MoSIFT	algoritmo para extraer la descripción de bajo nivel de una consulta de vídeo
MoviesFight	juego de datos de películas
Neocognitron	red neuronal artificial jerárquica de varias capas, considerado como el antepasado de la red neuronal convolucional.
output	salida
output nodes	nodos de salida
overfitting	sobreajuste
overlearning	sobre aprendizaje
panning	paneo (movimiento de cámara)
Perceptron	primera neurona artificial o unidad básica de inferencia de discriminador lineal.
plain network	red simple
pooling layers	capa de agrupación
Precision	Precisión del modelo, número de positivos entre el número total
Predicted	predicho
Python	lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código
Pytorch	biblioteca para el aprendizaje profundo de datos de entrada irregulares, como gráficos, nubes de puntos y múltiples.
Recall	Memoria, número de aciertos entre los verdaderos positivos más los falsos positivos
ResNet	red neuronal residual
Restricted Boltzmann Machine (Harmonium)	red neuronal artificial estocástica generativa que puede aprender una distribución de probabilidad sobre su conjunto de entradas.
RMSprop	algoritmo de optimización
RWF-2000	base de datos de vídeo abierta a gran escala para la detección de violencia
salience	prominencia o rasgo sobresaliente
shallow classifier	clasificador superficial
single frame	un solo cuadro
slow fusion	fusión lenta
SoftMax	función exponencial normalizada
Theano	*biblioteca de Python y un compilador de optimización para manipular y evaluar expresiones matemáticas, especialmente las de valor matricial.*
TimeDistributed	contenedor de Keras, capa muy útil para trabajar con datos de series temporales o cuadros de vídeo
transfer learning	transferencia de aprendizaje
True Negative	Verdadero negativo, cuando un caso fue negativo y se predijo positivo
True Positive	Verdadero positivo, cuando un caso fue positivo y se predijo positivo
uint_8	*Operadores unarios de tamaño 8*
underfitting	Infraajuste
vanishing gradient	gradiente desvaneciente
ViolentFlow	Juego de datos de violencia
Wordnet	base de datos léxica que agrupa palabras en inglés en conjuntos de sinónimos llamados *synsets*, proporcionando definiciones cortas y generales, y almacenando las relaciones semánticas entre los conjuntos de sinónimos
zoom	enfocar (ampliar-reducir imagen)

Explicación básica de las funciones de los módulos

Módulo de entrenamiento

Función	Descripción
frameLoader	a partir de una lista de fotogramas, carga en una matriz la información de estos fotogramas
cropDark	permite eliminar bordes negros de fotogramas si se encuentran presentes
dataAugmentation	de forma aleatoria, modifica mediante aumentos de cámara y traslaciones las imágenes de entrada
getSequence	obtiene a partir de una lista de fotogramas, una matriz con su información ya preparada para entrenamiento junto a su categoría
saveFrames	a partir de un vídeo, crea sus fotogramas correspondientes y devuelve su localización
getGenerator	a partir de una lista de fotogramas, devuelve un generador que irá devolviendo secuencias a medida que se requiera
createDataset	realiza un análisis inicial del *Dataset*, convirtiendo los vídeos en series de fotogramas y realizando la separación entre conjunto de entrenamiento, validación y prueba
getGenerator	a partir de los resultados de *createDataset*, crea un generador para los conjuntos de entrenamiento y validación, obtiene las secuencias del conjunto de prueba y los devuelve
trainModel	después de obtener el modelo por *createModel*, entrena el modelo y guarda el mejor resultado de éste, junto con su historial de entrenamiento
createModel	a partir de los parámetros especificados, crea el modelo a entrenar.
Main de entrenamiento	función principal donde se inicializan los parámetros del modelo y la localización de los *Datasets*, y donde se llama por cada Dataset a *getGenerators* y a *trainModel*
TestCallback	comprueba en cada época los resultados del modelo frente al conjunto de prueba
wrapperGenerator	Función que envuelve a un generador en otro, para poder preprocesar la información según lo requiere *ResNet*

Módulo de análisis

Función	Descripción
getGraph	a partir del histórico de entrenamiento, pinta el gráfico del histórico de entrenamiento
loadModel	carga el modelo preentrenado en el módulo de entrenamiento
evaluateImage	a partir de la imagen o imágenes que hayamos puesto para evaluar, evaluará la imagen y calculará el reporte de clasificación del modelo
getHeatmap	a partir del modelo preentrenado y la imagen a evaluar, calculará el mapa de calor de la imagen
Main de análisis	a partir de los resultados del módulo de entrenamiento, obtiene la gráfica del histórico de entrenamiento, y evalúa con el modelo entrenado las imágenes que se quieran evaluar, obteniendo el reporte de clasificación del modelo según las imágenes evaluadas, y el mapa de calor de estas
GradCAM	implementación del algoritmo *GradCAM* referenciada, modificada para su uso en secuencias de imágenes
grafica_entrenamiento	función que realiza la representación gráfica del histórico de entrenamiento, señalando la época con mayor tasa de prueba

Anexos

Código del módulo de entrenamiento

getGenerators (1): Módulo de entrenamiento

- **frameLoader**

```
def frameLoader(frames,figure_shape,to_norm = True):
 outputFrames = []
 for frame in frames:
  image = load_img(frame, target_size=(figure_shape,figure_shape),interpolation='bilinear')
  image_array = img_to_array(image)
  figure = (image_array / 255.).astype(np.float32)
  mean = [0.485, 0.456, 0.406]
  std = [0.229, 0.224, 0.225]
  figure = (figure - mean) / std
  outputFrames.append(figure)
 return outputFrames
```

- **cropDark**

```
def cropDark(image, crop_x, crop_y, x,y, figure_size):
    x_start = crop_x
    x_end = x-crop_x
    y_start = crop_y
    y_end = y-crop_y
    final_image = cv2.resize(image[y_start:y_end,x_start:x_end,:], (figure_size,figure_size))
    return final_image
```

- **dataAugmentation**

```
def dataAugmentation(sequence, percentage, use_cropping, figure_shape):
    rand = scipy.random.random()
    corners=["Up_Left","Up_Right","Down_Left", "Down_Right"]
    if rand > percentage:
        if(use_cropping):
            corner=random.choice(corners)
            for x_ in sequence:
                resize = int(figure_shape*scipy.random.random())
                if(corner =="Up_Left"):
                    x_start = 0
                    x_end = resize
                    y_start = 0
                    y_end = resize
                if (corner == "Down_Right"):
                    x start = figure shape-resize
                    x_end = figure_shape
                    y_start = figure_shape-resize
                    y_end = figure_shape
                if(corner =="Up_Right"):
                    x_start = 0
                    x end = resize
                    y_start = figure_shape-resize
                    y_end = figure_shape
                if (corner == "Down_Left"):
                    x_start = figure_shape-resize
                    x_end = figure_shape
                    y_start = 0
                    y_end = resize
                x_ = cv2.resize(x_[y_start:y_end,x_start:x_end, :], (figure_shape,
figure_shape)).astype(np.float32)
        sequence = [frame.transpose(3, 0, 1) for frame in sequence]
    return sequence
```

- **getSequence**

```
def getSequence(data_paths,labels,figure_shape,seq_length,classes=1, use_augmentation = False,
aug_crop =True,crop_dark = None):

    X, y = [], []
```

```
    seqLen = 0
    for dataPath, label in zip(data_paths,labels):
        frames = sorted(glob.glob(os.path.join(dataPath, '*jpg')))
        x = frameLoader(frames, figure_shape)

        if(crop_x_y):
        x = [img_aux.cropDark(x_,crop_x_y[0],crop_x_y[1],x_.shape[0],x_.shape[1],figure_shape) for
x_ in x]

        if use_augmentation:
        x = dataAugmentation(sequence, 0.6, aug_crop, figure_shape)

        X.append(x)
        y.append(label)
    X = pad_sequences(X, maxlen=seq_length, padding='pre', truncating='pre')
    return np.array(X), np.array(y)
```

- **saveFrames**

```
def saveFrames(path, filename, suffix, figures_path, skip_frames = 25, fix_len = None):
  sequenceLength = 0
  videoFiguresPath = os.path.join(figures_path, filename)
  if not os.path.exists(videoFiguresPath):
    os.makedirs(videoFiguresPath)
  videoFile = os.path.join(path, filename + suffix)
  label = 0

  videoCapture = cv2.VideoCapture(videoFile)

  if fix_len is not None:
    vidLen = int(videoCapture.get(cv2.CAP_PROP_FRAME_COUNT))
    skipFrames = int(float(vidLen)/float(fix_len))
  videoCapture.set(cv2.CAP_PROP_POS_MSEC, (sequenceLength * skipFrames))

  success, figure_ = videoCapture.read()
  success = True
  files = []
  while success:
    success, figure = videoCapture.read()
    if sequenceLength % skipFrames == 0:
        if success:
            currentFigure = figure
            imagePath = os.path.join(videoFiguresPath, "frame_%02d.jpg" % sequenceLength)
            files.append(imagePath)
            cv2.imwrite(imagePath, currentFigure)
    sequenceLength += 1
  video_images = dict(images_path = videoFiguresPath, name = filename, images_files = files,
sequence_length = sequenceLength, label = label)
  return video_images
```

- **getGenerator**

```
def
getGenerator(data_paths,labels,batch_size,figure_shape,seq_length,use_aug,use_crop,crop_x_y,classes
= 1):
     while True:
        indexes = np.arange(len(data_paths))
        np.random.shuffle(indexes)
        selectIndexes = indexes[:batch_size]
        dataPathsBatch = [data_paths[i] for i in selectIndexes]
        labelsBatch = [labels[i] for i in selectIndexes]
        X, y = getSequence(dataPathsBatch,labelsBatch,figure_shape,seq_length, classes,
use_augmentation = use_aug,aug_crop=use_crop,crop_dark=crop_x_y)
        yield X, y
```

- **createDataset**

```
def createDataset(datasets_video_path, figure_output_path,fix_len, force = False):
  videosListLength = []
  videosListFramesPath = []
  videosListLabels = []

  for dataset_name, dataset_video_path in datasets_video_path.items():
    dataset_figures_path = os.path.join(figure_output_path,dataset_name)
```

```
    if not os.path.exists(dataset_figures_path):
      os.makedirs(dataset_figures_path)

    videoListDict = []

    i = 0
    fin = len(os.listdir(dataset_video_path))

    for filename in os.listdir(dataset_video_path):
      i=i+1
      print("%s - %d/%d" % (filename, i, fin))
      if filename.endswith(".avi") or filename.endswith(".mpg"):

        summaryPath = os.path.join(dataset_figures_path,filename[:-4], 'summary.pkl')
        if os.path.isfile(summaryPath) and not force:
          with open(summaryPath, 'rb') as f:
            summary = pickle.load(f)
            print("Abriendo pickle...")

        else:
          summary = saveFrames(dataset_video_path, filename[:-4],filename[-4:],
dataset figures path, fix len=fix len)

          if dataset_name == "hocky": # Parseamos aquí el dataset para determinar su label
            if filename.startswith("fi"):
              summary['label'] = 1
          elif dataset_name == "violentflow":
            if "violence" in filename:
              summary['label'] = 1
          elif dataset_name == "movies":
            if "fi" in filename:
              summary['label'] = 1
          elif dataset_name == "rwf2000":
            if "FIGHT" in filename:
              summary['label'] = 1
          with open(summaryPath, 'wb') as f:
            pickle.dump(summary, f, pickle.HIGHEST_PROTOCOL) # Aquí es donde creamos el video
summary
          if os.path.isfile(summaryPath):
            print("Guardando pickle...")

        ## Una vez que tenemos el summary, nos guardamos la info en unas listas

        videosListLength.append(summary['sequence_length'])
        videosListFramesPath.append(summary['images_path'])
        videosListLabels.append(summary['label'])

  avgLength = int(float(sum(videosListLength)) / max(len(videosListLength), 1))


  trainPath, testPath, trainData, testData = train_test_split(videosListFramesPath,
videosListLabels, test_size=0.1, random_state=69)
  trainPath, validPath, trainData, validData = train_test_split(trainPath, trainData,
test_size=0.4, random_state=69)
  return trainPath, testPath, validPath, trainData, testData, validData, avgLength
```

- **getGenerators**

```
def getGenerators(dataset_name, dataset_videos, datasets_frames, fix_len, figure_size, force,
classes=1, use_aug=False, aug_crop=use_crop,crop_dark=crop_x_y, batch_size=2):
  train_path, test_path, valid_path, train_data, test_data, valid_data, avgLength =
dataset.createDataset(dataset_videos, datasets_frames, fix_len, force=force)

  if fix_len is not None:
    avgLength = fix_len

  cropXY = None
  if (crop_dark):
    cropXY = crop_dark[dataset_name]

  lenTrain, lenValid = len(train_path), len(valid_path)

  train_gen = dataGenerator(train_path, train_data, batch_size, figure_size, avgLength,
aug_crop=use_crop,crop_dark=crop_x_y, crop_x_y=cropXY, classes=classes)
  validate_gen = dataGenerator(valid_path, valid_data, batch_size, figure_size, avgLength,
aug_crop=False,crop_dark=crop_x_y, crop_x_y=cropXY, classes=classes)
```

```
  t = time.time()
  test_x, test_y = getSequences(test_path, test_data, figure_size, avgLength,
crop_x_y=cropXY,classes=classes)
  print("Generando secuencias de test... Hecho en " + str(time.time() - t) + " s")
  return train_gen, validate_gen, test_x, test_y, avgLength, lenTrain, lenValid, test_path
```

trainEvalNetwork (2): Módulo de entrenamiento

- **trainModel**

```
def trainModel(dataset_name, train_gen, validate_gen, test_x, test_y, seq_len, epochs, batch_size,
                      batch_epoch_ratio, initial_weights, size, cnn_arch, learning_rate,
                      optimizer, cnn_train_type, pre_weights, lstm_conf, len_train, len_valid,
dropout, classes,
                      patience_es=15, patience_lr=5):

    result = dict(dataset=dataset_name, cnn_train=cnn_train_type,
                  cnn=cnn_arch.__name__, lstm=lstm_conf[0].__name__, epochs=epochs,
                  learning_rate=learning_rate, batch_size=batch_size, dropout=dropout,
                  optimizer=optimizer[0].__name__, initial_weights=initial_weights,
seq_len=seq_len)

    model = createModel(size=size, seq_len=seq_len, learning_rate=learning_rate,
                                   optimizer_class=optimizer, initial_weights=initial_weights,
                                   cnn_class=cnn_arch, pre_weights=pre_weights,
lstm_conf=lstm_conf,
                                   cnn_train_type=cnn_train_type, dropout=dropout, classes=classes)

    res_path = "/content/tfg_eps/results/"
    mc = ModelCheckpoint('/content/best_model_' + dataset_name +
'.h5',monitor='val_acc',mode='auto',save_best_only=True, verbose=1)
    test_history = TestCallback((test_x, test_y))

    acum_tr_acc   = []
    acum_val_acc  = []
    acum_tr_loss  = []
    acum_val_loss = []

    best_i = 0

    history = model.fit_generator(
        steps_per_epoch=int((float(len_train) / float(batch_size * batch_epoch_ratio))/1),
        generator=wrapperGenerator(train_gen, preprocess_input),
        epochs=epochs,
        validation_data=wrapperGenerator(validate_gen, preprocess_input),
        validation_steps=int(float(len_valid) / float(batch_size)),
        callbacks=[EarlyStopping(monitor='val_loss', min_delta=0.001, patience=patience_es, ),
                   ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=patience_lr,
min_lr=1e-8, verbose=1),
                   test_history,
                   mc
                   ]
    )
    history to save = history.history
    history_to_save['test accuracy'] = test_history.test_acc
    history_to_save['test loss'] = test_history.test_loss

    model_name = ""
    for k, v in result.items():
        model_name = model_name + "_" + str(k) + "-" + str(v).replace(".", "d")
    model_path = os.path.join(res_path, model_name)
    pd.DataFrame(history_to_save).to_csv(model_path + "_train_results.csv")
    result['validation loss'] = min(history.history['val_loss'])
    result['validation accuracy'] = max(history.history['val_acc'])
    result['last validation loss'] = history.history['val_loss'][-1]
    result['last validation accuracy'] = history.history['val_acc'][-1]

    result['train accuracy'] = max(history.history['acc'])
    result['train loss'] = min(history.history['loss'])
    result['last train accuracy'] = history.history['acc'][-1]
    result['last train loss'] = history.history['loss'][-1]

    result['test accuracy'] = max(test_history.test_acc)
    result['test loss'] = min(test_history.test_loss)
    result['last test accuracy'] = test_history.test_acc[-1]
```

```
    result['last test loss'] = test_history.test_loss[-1]

    result['final lr'] = history.history['lr'][-1]
    result['total epochs'] = len(history.history['lr'])

    return result, model
```

- **createModel**

```
def createModel(size, seq_len ,learning_rate, optimizer_class, initial_weights, cnn_class,
pre_weights, lstm_conf, cnn_train_type, classes = 1, dropout = 0.0):
    input_layer = Input(shape=(seq_len, size, size, 3))
    if(cnn_train_type!='train'):
        if cnn_class.__name__ == "ResNet50V2":
            cnn = cnn_class(weights=pre_weights, include_top=False,input_shape =(size, size, 3))
        else:
            cnn = cnn_class(weights=pre_weights,include_top=False)
    else:
        cnn = cnn_class(include_top=False)

    if(cnn train type=='static'):
        for layer in cnn.layers:
            layer.trainable = False
    if(cnn_train_type=='retrain'):
        for layer in cnn.layers:
            layer.trainable = True

    cnn = TimeDistributed(cnn)(input_layer)
    lstm = lstm_conf[0](**lstm_conf[1])(cnn)
    lstm = MaxPooling2D(pool_size=(2, 2))(lstm)
    flat = Flatten()(lstm)

    flat = BatchNormalization()(flat,training=False)
    flat = Dropout(dropout)(flat)
    linear = Dense(1000,kernel_regularizer=l1l2(0.0001))(flat)

    relu = Activation('relu')(linear)
    linear = Dense(256)(relu)
    linear = Dropout(dropout)(linear)
    relu = Activation('relu')(linear)
    linear = Dense(10)(relu)
    linear = Dropout(dropout)(linear)
    relu = Activation('relu')(linear)

    activation = 'sigmoid'
    loss_func = 'binary_crossentropy'

    if classes > 1:
        activation = 'softmax'
        loss_func = 'categorical_crossentropy'
    predictions = Dense(classes,  activation=activation)(relu)

    model = Model(inputs=input_layer, outputs=predictions)
    optimizer = optimizer_class[0](lr=learning_rate, **optimizer_class[1])
    model.compile(optimizer=optimizer, loss=loss_func,metrics=['acc'])

    print(model.summary())
    display(SVG(model_to_dot(model, dpi=72).create(prog='dot', format='svg')))
    return model
```

main: función principal módulo de entrenamiento

```
def main():
    import scipy
    import cv2
    import pickle
    import glob
    import numpy as np
    import pandas as pd
    from keras.preprocessing.image import load_img, img_to_array
    from keras.preprocessing.sequence import pad_sequences
    from keras.utils import to_categorical
    from keras.applications.resnet_v2 import ResNet50V2, ResNet152V2
    from keras.applications.inception_v3 import InceptionV3
    from keras.applications.inception_resnet_v2 import InceptionResNetV2
```

```
    from keras.optimizers import RMSprop
    from keras.layers import LSTM, ConvLSTM2D
    from sklearn.model_selection import train_test_split
    from collections import defaultdict
    from keras.preprocessing import image
    import random

    from tensorflow.random import set_seed
    import matplotlib.pyplot as plt

    import sys

    datasets_videos = dict(
    hocky=dict(hocky="data/raw_videos/HockeyFights")
    violentflow=dict(violentflow="data/raw_videos/violentflow")
    movies=dict(movies="data/raw_videos/movies")
    rwf2000=dict(rwf2000="data/raw_videos/rwf2000")
    )

    crop_dark = dict(
    hocky=(11, 38)
    violentflow=None
    movies=None
    rwf2000=None
    )

    results = []
    res path = "results"
    cnn_arch = ResNet50V2
    learning_rate = 2e-5
    optimizer = (RMSprop, {})
    cnn_train_type = 'retrain'
    dropout = 0.1
    use aug = True
    fix_len = 20
    use_crop = True
    datasetsFrames = "data/raw_frames"
    figure_size = 244
    split_ratio = 0.1
    batch_size = 2
    batch_epoch_ratio = 0.5
    weights = 'imagenet'
    force = False
    lstm = (ConvLSTM2D, dict(filters=256, kernel_size=(3, 3), padding='same',
return_sequences=False))
    classes = 1
    epochs = 30

    for dsName, dsVideoPath in datasets_videos.items():
        train_gen, validate_gen, test_x, test_y, seq_len, len_train, len_valid,test_path =
getGenerators(dsName, dsVideoPath, datasetsFrames, fix_len, figure_size, force, classes, use_aug,
use_crop, crop_dark, batch_size)
        print("getGenerators terminado.")
        result, model = model.trainModel(epochs=epochs, dataset_name=dsName, train_gen=train_gen,
validate_gen=validate_gen,test_x=test_x, test_y=test_y, seq_len=seq_len,
batch_size=batch_size,batch_epoch_ratio=0.5, initial_weights=initial_weights,
size=figure_size,cnn_arch=cnn_arch, learning_rate=learning_rate,optimizer=optimizer,
cnn_train_type=cnn_train_type,pre_weights=weights, lstm_conf=lstm, len_train=len_train,
len_valid=len_valid,dropout=dropout, classes=classes)
```

Funciones auxiliaries: Módulo de entrenamiento

```
class TestCallback(Callback):
    def __init__(self, test_data):
        self.test_data = test_data
        self.test_loss = []
        self.test acc = []

    def on_epoch_end(self, epoch, logs={}):
        x, y = self.test_data
        loss, acc = self.model.evaluate(preprocess_input(x), y, batch_size=2, verbose=0)
        self.test_loss.append(loss)
        self.test acc.append(acc)
        print('\nTesting loss: {}, acc: {}\n'.format(loss, acc))
```

```
from matplotlib.ticker import MaxNLocator

def wrapperGenerator(dataGenerator, preprocess):
    while True:
        x, y = next(dataGenerator)
        x = preprocess(x)
        yield(x,y)
```

Código del módulo de análisis

- **getGraph**

```
def getGraph():
    import pandas as pd
    import numpy as np
    ds = pd.read_csv("dataset.csv",",")

    import numpy as np
    from matplotlib.ticker import MaxNLocator
    best_i = np.argmax(ds['test accuracy'])
    grafica_entrenamiento(ds['acc'],ds['val_acc'],ds['test
accuracy'],ds['loss'],ds['val_loss'],ds['test loss'], best_i)
```

- **loadModel**

```
def loadModel():
    return load_model("model.h5")
```

- **evaluateImage**

```
def evaluateImage(mod):

    datasets_videos = dict(
    hocky=dict(hocky="data/raw_videos/HockeyFights")
    violentflow=dict(violentflow="data/raw_videos/violentflow")
    movies=dict(movies="data/raw_videos/movies")
    rwf2000=dict(rwf2000="data/raw_videos/rwf2000")
    )

    crop_dark = dict(
    hocky=(11, 38)
    violentflow=None
    movies=None
    rwf2000=None
    )

    results = []
    res_path = "results"
    cnn_arch = ResNet50V2
    learning_rate = 2e-5
    optimizer = (RMSprop, {})
    cnn_train_type = 'retrain'
    dropout = 0.1
    use_aug = True
    fix_len = 20
    use_crop = True
    datasetsFrames = "data/raw_frames"
    figure_size = 244
    split_ratio = 0.1
    batch_size = 2
    batch_epoch_ratio = 0.5
    weights = 'imagenet'
    force = False
    lstm = (ConvLSTM2D, dict(filters=256, kernel_size=(3, 3), padding='same',
return_sequences=False))
    classes = 1
    epochs = 30

    for dsName, dsVideoPath in datasets_videos.items():
        train_gen, validate_gen, test_x, test_y, seq_len, len_train, len_valid,test_path =
getGenerators(dsName, dsVideoPath, datasetsFrames, fix_len, figure_size, force, classes, use_aug,
use_crop, crop_dark, batch_size)
        test_pred = mod.predict(test_x, batch_size=1, verbose=1)
```

```
        prediction = test_pred > 0.5
        prediction2 = int(prediction == 'True')
        print(classification_report(test_y[:len(prediction)], prediction))

        df = pd.DataFrame()
        df["pred"] = test_pred[0]
        df["label"] = test_y
        pd.set_option('display.max_columns', None)
        pd.set_option('display.max_rows', 2000)
        pd.set_option('display.float_format', '{:.3f}'.format)
        display(df)
        df.sort_values(by=['pred'])
        results.append(test_x)
    return results
```

- **getHeatmap**

```
def getHeatmap(image, mod):
    cam = GradCAM(mod, 0)
    heatmap = cam.compute_heatmap(image)
    img array = []

    for i in range(0,len(image)):
        img3_original_alpha = image[i]
        img4_original_alpha = cv2.normalize(src=img3_original_alpha, dst=None, alpha=0, beta=255,
norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
        heatmap alpha = heatmap[i]
        (heatmap_beta, output) = cam.overlay_heatmap(heatmap_alpha, img4_original_alpha, alpha=0.5)
        height, width, layers = img4_original_alpha.shape
        size = (height,width)
        img_array.append(output)

        out = cv2.VideoWriter('test.avi',cv2.VideoWriter fourcc(*'DIVX'), 20, size)
        for i in range(len(img_array)):
            out.write(img_array[i])
        out.release()
```

main: función principal módulo de análisis

```
def main():
    import matplotlib.pyplot as plt
    import os
    import pickle
    import cv2
    import glob
    import numpy as np
    import pandas as pd
    import argparse
    import imutils
    import tensorflow as tf

    from pyimagesearch.gradcam import GradCAM
    from google.colab.patches import cv2_imshow
    from tensorflow.keras.models import Model,load_model
    from tensorflow.keras.applications.resnet50 import preprocess_input
    from keras.preprocessing.sequence import pad_sequences
    from keras.preprocessing.image import load_img, img_to_array

    getGraph()
    mod = loadModel()
    images = evaluateImage(mod)
    [getHeatmap(image,mod) for image in images]
```

Funciones auxiliares: Módulo de análisis

```
class GradCAM:
       def   init  (self, model, classIdx, layerName=None):
              self.model = model
              self.classIdx = classIdx
              self.layerName = layerName

              if self.layerName is None:
```

```
                        self.layerName = self.find_target_layer()

        def find_target_layer(self):
                for layer in reversed(self.model.layers):
                        if len(layer.output_shape) == 4:
                                return layer.name

                raise ValueError("Could not find 4D layer. Cannot apply GradCAM.")

        def compute_heatmap(self, image, eps=1e-8):
                gradModel = Model(
                        inputs=[self.model.inputs],
                        outputs=[self.model.get_layer('time_distributed_1').output,
                                self.model.output])

                with tf.GradientTape() as tape:
                        inputs = tf.cast(image, tf.float32)
                        (convOutputs, predictions) = gradModel(inputs)
                        loss = predictions[:, self.classIdx]
                        #print("is this loss? "+ str(loss))

                grads = tape.gradient(loss, convOutputs)

                castConvOutputs = tf.cast(convOutputs > 0, "float32")
                castGrads = tf.cast(grads > 0, "float32")
                guidedGrads = castConvOutputs * castGrads * grads

                convOutputs = convOutputs[0]
                guidedGrads = guidedGrads[0]

                heatmap_final = []
                for i in range(0,len(convOutputs)):
                        conv = convOutputs[i]
                        guided = guidedGrads[i]

                        weights = tf.reduce_mean(guided, axis=(0,1))
                        cam = tf.reduce_sum(tf.multiply(weights, conv), axis=-1)

                        (w, h) = (image.shape[3], image.shape[2])
                        heatmap = cv2.resize(cam.numpy(), (w, h))

                        numer = heatmap - np.min(heatmap)
                        denom = (heatmap.max() - heatmap.min()) + eps
                        heatmap = numer / denom
                        heatmap = (heatmap * 255).astype("uint8")

                        heatmap_final.append(heatmap)
                return np.asarray(heatmap_final)

        def overlay_heatmap(self, heatmap, image, alpha=0.5,
                colormap=cv2.COLORMAP_VIRIDIS):
                heatmap = cv2.applyColorMap(heatmap, colormap)
                output = cv2.addWeighted(image, alpha, heatmap, 1 - alpha, 0)
                return (heatmap, output)

def grafica_entrenamiento(tr_acc, val_acc, test_acc, tr_loss, val_loss, test_loss, best_i,
                          figsize=(20,10)):
    plt.figure(figsize=figsize)
    ax = plt.subplot(1,2,1)
    #ax.set_xlim([0,1])
    #ax.set_ylim([0,1])
    plt.plot(1+np.arange(len(tr_acc)),  100*np.array(tr_acc))
    plt.plot(1+np.arange(len(val_acc)), 100*np.array(val_acc))
    plt.plot(1+np.arange(len(test_acc)), 100*np.array(test_acc))
    plt.plot(1+best_i, 100*test_acc[best_i], 'or')
    plt.title('tasa de acierto del modelo (%)', fontsize=18)
    plt.ylabel('tasa de acierto (%)', fontsize=18)
    plt.xlabel('época', fontsize=18)
    plt.legend(['entrenamiento', 'validación', 'test'], loc='upper left')
    ax.xaxis.set_major_locator(MaxNLocator(integer=True))

    ax2 = plt.subplot(1,2,2)
    #ax2.set_xlim([0,1])
    #ax2.set_ylim([0,1])
    plt.plot(1+np.arange(len(tr_acc)), np.array(tr_loss))
    plt.plot(1+np.arange(len(val_acc)), np.array(val_loss))
```

```
    plt.plot(1+np.arange(len(test_acc)), np.array(test_loss))
    plt.plot(1+best_i, test_loss[best_i], 'or')
    plt.title('loss del modelo', fontsize=18)
    plt.ylabel('loss', fontsize=18)
    plt.xlabel('época', fontsize=18)
    plt.legend(['entrenamiento', 'validación','test'], loc='upper left')
    ax.xaxis.set_major_locator(MaxNLocator(integer=True))


    plt.show()
```

[1] Zhang,W., Xue, X., Sun,Z., Guo, Y., ChiM., Lu,H. (2007) *Efficient Feature Extraction for Image Classification.* IEEE 11th International Conference on Computer Vision, Rio de Janeiro, pp. 1-8, doi: 10.1109/ICCV.2007.4409058.
https://ieeexplore.ieee.org/document/4409058

[2] Wikipedia (2020) *Haar-like feature*. Last edited on 06 May 2020
https://en.wikipedia.org/wiki/Haar-like_feature
Último acceso: 2020/06/10

[3] Saha, S., Singh, G., Sapienza, M., Torr, P., Cuzzolin, F. (2016). *Deep Learning for Detecting Multiple Space-Time Action Tubes in Videos*. 58.1-58.13. 10.5244/C.30.58.
https://arxiv.org/pdf/1608.01529.pdf

[4] Dalal, N., Triggs, B., (2005). *Histograms of Oriented Gradients for Human Detection*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005).
https://www.researchgate.net/publication/281327886_Histograms_of_Oriented_Gradients_for_Human_Detection/citation/download

[5] Chaudhry, R., Ravichandran, A., Hager, G., Vidal, R., (2009). *Histograms of oriented optical flow and Binet-Cauchy kernels on nonlinear dynamical systems for the recognition of human actions.* 1932-1939.
https://www.researchgate.net/publication/221362187_Histograms_of_oriented_optical_flow_and_Binet-Cauchy_kernels_on_nonlinear_dynamical_systems_for_the_recognition_of_human_actions

[6] Pietikäinen, M. (2005) *Image Analysis with Local Binary Patterns*
https://www.researchgate.net/publication/220809552_Image_Analysis_with_Local_Binary_Patterns

[7] Peng, X., Qiao, Y., Peng. Q,. Xianbiao Q (2013). *Exploring Motion Boundary based Sampling and Spatial-Temporal Context Descriptors for Action Recognition*
http://www.bmva.org/bmvc/2013/Papers/paper0059/paper0059.pdf

[8] Poppe, R. (2010) *A survey on vision-based human action recognition*. Image and Vision Computing 28, 976–990
https://www.sciencedirect.com/science/article/abs/pii/S0262885609002704

[9] Giannakopoulos, T., Pikrakis, A., & Theodoridis, S. (2010). *A multimodal approach to violence detection in video sharing sites.* In Pattern Recognition (ICPR), 2010 20th International Conference on (pp. 3244-3247). IEEE.
https://www.researchgate.net/publication/220931681_A_Multimodal_Approach_to_Violence_Detection_in_Video_Sharing_Sites

[10] Nievas, E. B., Suarez, O. D., García, G. B., Sukthankar, R. (2011). *Violence detection in video using computer vision techniques.* In Computer Analysis of Images and Patterns (pp. 332-339). Springer Berlin Heidelberg.
https://link.springer.com/chapter/10.1007/978-3-642-23678-5_39

[11] Ming-yu, C., Hauptmann., A. (2018) *Mosift: Recognizing human actions in surveillance videos*. (2009).
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.962.7080&rep=rep1&type=pdf

[12] Eyben, F., Weninger, F., Lehment, N., Schuller, B., Rigoll, G. (2013*). Affective video retrieval: Violence detection in Hollywood movies by large-scale segmental feature extraction.* PloS one, 8(12), e78506.
https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0078506

[13] Mironică, I., Duta, I, Ionescu, B., Sebe, N. (2015). *A modified vector of locally aggregated descriptors approach for fast video classification.* Multimedia Tools and Applications. 75. 10.1007/s11042-015-2819-7
https://link.springer.com/article/10.1007/s11042-015-2819-7

[14] Dehe, Y., Jingguo, L., Danlu, Z., Jingfa, Z., Jing., Y. (2019) *Extracting multi-features and optimizing feature space with sparse auto-encoder over WorldView-2 images*, International Journal of Remote Sensing, 40:16, 6418-6443, DOI: 10.1080/01431161.2019.1594431
https://www.tandfonline.com/doi/abs/10.1080/01431161.2019.1594431?journalCode=tres20&

[15] Voulodimos, A., Doulamis, N., Doulamis, A., Protopapadakis, E. (2018) *Deep Learning for Computer Vision: A Brief Review*
https://www.hindawi.com/journals/cin/2018/7068349/

[16] Rumelhart, D. E., Hinton, G. E., y Williams, R. J. (1986) *Learning representations by back-propagating errors*. Nature 323, 533–536.

https://www.nature.com/articles/323533a0
[17] Kostadinow, S. (2019) *Understanding Backpropagation Algorithm*
https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd
[18] LeCun, Y., Boser, B., Denker,J., Henderson, D. Howard, R., Hubbard,W., Jackel, L. (1989) *Backpropagation Applied to Handwritten Zip Code Recognition*. Neural Computation, vol. 1, no. 4, pp. 541-551, Dec. 1989, doi: 10.1162/neco.4.541.
https://ieeexplore.ieee.org/document/6795724
[19] LeCun, Y., Bottoy, L., Bengio, Y., Haffner. P., (1998) *Gradient-based learning Applied to Document Recognition.* IEEE International Conference
http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
[20] Fang, S. (2020) *Review of LeNet-5: How to design the architecture of CNN*. Towards Data Science.
https://towardsdatascience.com/review-of-lenet-5-how-to-design-the-architecture-of-cnn-8ee92ff760ac
Último acceso: 2020/06/10
[21] Poznanski, A., Wolf, L.(2016) *CNN-N-Gram for Handwriting Word Recognition*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
https://ieeexplore.ieee.org/document/7780622
[22] Olah, C. (2015) *Understanding LSTM Networks*
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Último acceso: 2020/06/10
[23] Hochreiter, S., Schmidhuber, J. (1997). *Long Short-term Memory*. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.
https://www.researchgate.net/publication/13853244_Long_Short-term_Memory
[24] Misra, A. (2019) *Using RNNs for Machine Translation*. Towards data science.
https://towardsdatascience.com/using-rnns-for-machine-translation-11ddded78ddf
Último acceso: 2020/06/10
[25] Sotner, D. Müller, L. (2013) *Application of LSTM Neural Networks in Language Modelling*. International Conference on Text, Speech and Dialogue
https://link.springer.com/chapter/10.1007/978-3-642-40585-3_14
[26] Issa, A. (2019) *Generating Original Classical Music with an LSTM Neural Network and Attention*. Medium
https://medium.com/@alexissa122/generating-original-classical-music-with-an-lstm-neural-network-and-attention-abf03f9ddcb4
[27] Nayak, M. (2018) *An Intuitive Introduction of Boltzmann Machine*
https://medium.com/datadriveninvestor/an-intuitive-introduction-of-boltzmann-machine-8ec54980d789
Último acceso: 2020/06/10
[28] Duran, J. (2019) *Técnicas de Regularización Básicas para Redes Neuronales.* MetaDatos
https://medium.com/metadatos/técnicas-de-regularización-básicas-para-redes-neuronales-b48f396924d4
[29] Fischer, A., Igel, C. (2012) *An Introduction to Restricted Boltzmann Machines*. Iberoamerican Congress on Pattern Recognition
https://link.springer.com/chapter/10.1007/978-3-642-33275-3_2
[30] Hinton, G., Osindero, S., The, Y., (2006) *A fast learning algorithm for deep belief nets.* Neutral Computation Vol 18, 7
https://dl.acm.org/doi/10.1162/neco.2006.18.7.1527
[31] Salakhutdinov, R.. Hinton, G. (2009) *Deep Boltzmann Machines.* Department of Computer Science. University of Toronto
http://www.cs.toronto.edu/~fritz/absps/dbm.pdf
[32] Bladi, P. (2012) *Autoencoders, Unsupervised Learning, and Deep Architectures.* JMLR: Workshop and Conference Proceedings 27:37–50
http://proceedings.mlr.press/v27/baldi12a/baldi12a.pdf
[33] Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P. (2008) *Extracting and Composing Robust Features with Denoising Autoencoders.* CS. Toronto.
https://www.cs.toronto.edu/~larocheh/publications/icml-2008-denoising-autoencoders.pdf
[34] Krizhevsky, A. Sutskeve, I. y Hinton, G. (2012) *ImageNet Classification with Deep Convolutional Neural Networks.*
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[35] Llamas, J., Lerones, P., Medina, R., Zalama, E., y Gómez, J. (2017). *Classification of Architectural Heritage Images Using Deep Learning Techniques*. Applied Sciences. 7. 992. 10.3390/app7100992. https://www.mdpi.com/2076-3417/7/10/992/htm

[36] Chellapilla, K., Puri, S., Simard, P. (2006) *High Performance Convolutional Neural Networks for Document Processing*. Tenth International Workshop on Frontiers in Handwriting Recognition, Université de Rennes 1, Oct 2006, La Baule (France). ffinria-00112631f https://hal.inria.fr/file/index/docid/112631/filename/p1038112283956.pdf

[37] Cires, D., Meier, D. Masci, J., Gambardella, L., Schmidhuber. J.(2011) *Flexible, High Performance Convolutional.* Neural Networks for Image Classification. IDSIA, USI and SUPSI http://people.idsia.ch/~juergen/ijcai2011.pdf

[38] Fukushima, K., (1980) *Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position.* Biol. Cybernetics 36, 193 202 https://www.rctn.org/bruno/public/papers/Fukushima1980.pdf

[39] Weng, J., Ahuja, N., Huang, T (1993). *Learning recognition and segmentation of 3D objects from 2D images*. Proc. 4th International Conf. Computer Vision: 121–128. https://www.cse.msu.edu/~weng/research/CresceptronICCV1993.pdf

[40] He, K, Zhang, X., Ren S. and Sun J.(2015) *Deep Residual Learning for Image Recognition.* Cornell University. NY. https://arxiv.org/abs/1512.03385

[41] Wikipedia (2020) *Machine learning*. Last edited on 29 May 2020 https://en.wikipedia.org/wiki/Machine_learning Último acceso: 2020/06/10

[42] Mitchell, T. (1997) *Machine Learning.* McGraw Hill http://www.cs.cmu.edu/~tom/mlbook.html

[43] Sunil R. (2017) *Commonly used Machine Learning Algorithms (with Python and R Codes*). Analytics Vidhya. https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/ Último acceso: 2020/06/10

[44] Gonzalez, L. (2019) *Árboles de Decisión Clasificación – Teoría* https://ligdigonzalez.com/arboles-de-decision-clasificacion-teoria-machine-learning/ Último acceso: 2020/06/10

[45] Swaminathan, S. (2018) *Logistic Regression — Detailed Overview*. Towards Data Science https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc Último acceso: 2020/06/10

[46] Patel, S. (2017) *Chapter 2: SVM (Support Vector Machine) — Theory*. Medium https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72 Último acceso: 2020/06/10

[47] Brownlee, J. (2019) *A Gentle Introduction to Bayesian Belief Networks.* Machine Learning Mastery https://machinelearningmastery.com/introduction-to-bayesian-belief-networks/ Último acceso: 2020/06/10

[48] Chandra, A. (2018) *Perceptron: The Artificial Neuron (An essential upgrade to the McCulloch-Pitts neuron).* Towards Data Science https://towardsdatascience.com/perceptron-the-artificial-neuron-4d8c70d5cc8d Último acceso: 2020/06/10

[49] FD (2017) *Batch normalization in Neural Networks.* Towards Data Science https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c Último acceso: 2020/06/10

[50] Sinnott, R., Sun, Y. (2016) *A Case Study in Big Data Analytics*. Big Data https://www.sciencedirect.com/topics/computer-science/sigmoid-function Último acceso: 2020/06/10

[51] Polamuri, S., (2017) *Difference Between Softmax Function and Sigmoid Function* . Dataaspitant https://dataaspirant.com/2017/03/07/difference-between-softmax-function-and-sigmoid-function/

[52] Brownlee, J. (2019) *A Gentle Introduction to the Rectified Linear Unit (ReLU).* Machine Learning Mastery. https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/ Último acceso: 2020/06/10

[53] Alberti, M. (2018) *Redes neuronales capsulares.* Deep Learning – ES

https://www.deeplearningitalia.com/redes-neuronales-capsulares/
Último acceso: 2020/06/10
[54] Zhang, L., Hong, L., King, X. (2019) *Evolving feedforward artificial neural networks using a two-stage approach*. Neurocomputing Volume 360, 30 September, Pages 25-36
https://www.sciencedirect.com/science/article/pii/S0925231219309191
[55] Sherstinsky, A. (2020) *Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network*. Cornell University
https://arxiv.org/abs/1808.03314
[56] Larrañaga, P., Inza I. y Moujahid, A. (2020) *Redes Neuronales*. Universidad del País Vasco
http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/
[57] Smart Panel (2020) *¿Qué es el Deep Learning?*
https://www.smartpanel.com/que-es-deep-learning/
Último acceso: 2020/06/10
[58] Pihlajamäki, T. (2020) *Multi-resolution Short-time Fourier Transform Implementation of Directional Audio Coding*
https://www.researchgate.net/figure/The-calculation-of-convolution-Two-signals-red-and-blue-are-convolved-with-each_fig10_267239829
[59] Cowley J. (2018) *Redes neuronales convolucionales. Utilizar Python para implementar una red sencilla que clasifica dígitos escritos a mano*. IBM
https://www.ibm.com/developerworks/ssa/library/cc-convolutional-neural-network-vision-recognition/index.html
[60] Sinhal, K. and Sachan, A. (2020) *Understand Resnet / AlexNet / Vgg / Inception.*
https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/
Último acceso: 2020/06/10
[61] Katarzyna, J., Wojciech, M., (2017) *On Loss Functions for Deep Neural Networks in Classification*. Theoretical Foundations of Machine Learning 2017
https://arxiv.org/pdf/1702.05659.pdf
[62] He, K, Zhang, X., Ren S. and Sun J. (2016) *Identity Mappings in Deep Residual Networks.* Cornell University. NY.
https://arxiv.org/abs/1603.05027
[63] Szegedy, C. (2014) *Building a deeper understanding of images*. Google Al
https://ai.googleblog.com/2014/09/building-deeper-understanding-of-images.html
[64] Cao, X. Wipf, D., Wen,F. Duan,G.and Sun,J. (2013) *A practical transfer learning algorithm for face verification*," in Proceedings of the 14th IEEE International Conference
https://www.microsoft.com/en-us/research/wp-content/uploads/2013/01/TransferLearning.pdf
[65] Lawrence, S. Giles, C. Tsoi, A., and Back, A. (1997) *Face recognition: a convolutional neural-network approach*, IEEE Transactions on Neural Networks and Learning Systems, vol. 8,no. 1, pp. 98–113.
https://dl.acm.org/doi/10.1109/72.554195
[66] Schroff, F., Kalenichenko, D. Philbin, J, (2015) *FaceNet: A Unified Embedding for Face Recognition and Clustering*. Cornell University
https://arxiv.org/abs/1503.03832
[67] Taigman, Y., Yang, M., Ranzato M., Wolf, L. (2014) *DeepFace: Closing the Gap to Human-Level Performance in Face Verification*, IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 1701-1708, doi: 10.1109/CVPR.2014.220.
https://ieeexplore.ieee.org/document/6909616
[68] Gan, C., Wang, N., Yang, Y., Yeung, D., Hauptmann, A., *DevNet: A Deep Event Network for multimedia event detection and evidence recounting 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, 2015, pp. 2568-2577, doi: 10.1109/CVPR.2015.7298872.
https://ieeexplore.ieee.org/document/7298872
[69] Kautz, T., Groh, B., Hannink, J. Jensen, U., Strubberg, H., Eskofier, B. (2017) *Activity recognition in beach volleyball using a DEEp Convolutional Neural NETwork: leveraging the potential of DEEp Learning in sports*," Data Mining and Knowledge. Discovery, vol. 31, no. 6, pp. 1678–1705.
https://dl.acm.org/doi/10.1007/s10618-017-0495-0
[70] Ronao, C., Cho, S. (2016) *Human activity recognition with smartphone sensors using deep learning neural networks*. Expert Systems with Applications, vol. 59, pp. 235–244
https://www.sciencedirect.com/science/article/abs/pii/S0957417416302056

[71] Kitsikidis, A., Dimitropoulos, K., Douka, S., Grammalidis, N. (2014). *Dance analysis using multiple Kinect sensors*. VISAPP- Proceedings of the 9th International Conference on Computer Vision Theory and Applications. 2. 789-795.
https://www.researchgate.net/publication/287882481_Dance_analysis_using_multiple_kinect_sensors/citation/download

[72] Felzenszwalb, P.F., Huttenlocher, D.P. (2005). *Pictorial Structures for Object Recognition*. International Journal of Computer Vision 61, 55–79
https://doi.org/10.1023/B:VISI.0000042934.15159.49

[73] LISA Lab (2018) *Convolutional Neural Networks (LeNet).* DeepLearning 0.1 docs.
http://deeplearning.net/tutorial/lenet.html
Último acceso: 2020/06/10

[74] Karpathy, A., Toderici, G., Shetty, S., Leung,T., Sukthankar, R. Fei-Fei, L. (2014) *Large-scale Video Classification with Convolutional Neural Networ*ks. Proceedings of International Computer Vision and Pattern Recognition, IEEE
https://research.google/pubs/pub42455/

[75] Simonyan, K., Zisserman, A., (2014) *Two-Stream Convolutional Networks for Action Recognition in Videos.* Cornell University
https://arxiv.org/abs/1406.2199

[76] Eskinder, A., Sanjay., D. (2016). *Motion Estimation in Video Coding using Simplified Optical Flow Technique*. International Arab Journal of Information Technology. 13. 770-776.
https://www.researchgate.net/publication/316345345_Motion_Estimation_in_Video_Coding_using_Simplified_Optical_Flow_Technique/citation/download

[77] Warren, W. (2010). *Optic Flow*. 10.1016/B978-012370880-9.00311-X.
https://www.researchgate.net/publication/286038595_Optic_Flow/citation/download

[78] Tran, D., Bourdev, L., Fergus, R., Torresani, L., Paluri, M. (2015) L*earning Spatiotemporal Features with 3D Convolutional Networks*
https://arxiv.org/pdf/1412.0767.pdf

[79] Sudhakaran, S., Lanz, O., (2017) *Learning to Detect Violent Videos using Convolutional Long Short-Term Memory*
https://arxiv.org/pdf/1709.06531.pdf

[80] Ashikin, A., Norhalina., S (2016). *A Review on Violence Video Classification Using Convolutional Neural Networks*. 10.1007/978-3-319-51281-5_14.
https://www.researchgate.net/publication/312031821_A_Review_on_Violence_Video_Classification_Using_Convolutional_Neural_Networks

[81] Fath U Min Ullah, F., Ullah, A. Muhammad, K., Ul, I., Baik, S. (2019) *Violence Detection Using Spatiotemporal Features with 3D Convolutional Neural Network*
https://doi.org/10.3390/s19112472

[82] M. Ramzan, M. (2019) *A Review on State-of-the-Art Violence Detection Techniques*. IEEE Access, vol. 7, pp. 107560-107575, 2019, doi: 10.1109/ACCESS.2019.2932114.
https://ieeexplore.ieee.org/abstract/document/8782115

[83] Li. C., Zhu, L., Zhu, D., Chen, J., Pan, Z., Li, X., Wang, B. (2018) *End-to-end Multiplayer Violence Detection based on Deep 3D CNN*
https://dl.acm.org/doi/10.1145/3301326.3301367

[84] Zhou, B., Khosla, A., Lapedriza, A., Oliva, A, Torralba, A. (2016) *Learning Deep Features for Discriminative Localization*. IEEE
http://cnnlocalization.csail.mit.edu/Zhou_Learning_Deep_Features_CVPR_2016_paper.pdf

[85] Huang, G., Liu, Z., Van der Maatern, L., Weinberger, Q., (2016) *Densely Connected Convolutional Networks*. Cornell University
https://arxiv.org/abs/1608.06993

[86] GitHub (2020) *Deep Residual Learning for Image Recognition*
https://github.com/keras-team/keras-applications/blob/master/keras_applications/resnet50.py
Último acceso: 2020/06/10

[87] *Python Docs: Functional Programming HOWTO*
https://docs.python.org/3/howto/functional.html
Último acceso: 2020/06/10

[88] Shorten, C., Khoshgoftaar, T. (2019) *A survey on Image Data Augmentation for Deep Learning*. Big Data.
https://link.springer.com/content/pdf/10.1186/s40537-019-0197-0.pdf

[89] Srivastava, N., Hinton, G., Krizhevsky, A. Sutskever, I., Sutskever R. (2014) *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Journal of Machine Learning Research 15. 1929-1958
https://pdfs.semanticscholar.org/6c8b/30f63f265c32e26d999aa1fef5286b8308ad.pdf?_ga=2.205586579.1807662726.1591718751-172242360.1591273965
[90] Keras (2020) *ResNet and ResNetV2*
https://keras.io/api/applications/resnet/#resnet-and-resnetv2
Último acceso: 2020/06/10
[91] TensorFlow (2020) *An end-to-end open source machine learning platform*
https://www.tensorflow.org/
Último acceso: 2020/06/10
[92] Wikipedia (2020) *Theano.* Last edited on 14 September 2012
https://en.wikipedia.org/wiki/Theano
Último acceso: 2020/06/10
[93] Wikipedia (2020) *PyTorch*. Last edited on 29 April 2020
https://en.wikipedia.org/wiki/PyTorch
Último acceso: 2020/06/10
[94] Blunsden, S., Fisher, R., (2009) *The behave video dataset: ground truthed video for multi-person behavior classification*
http://homepages.inf.ed.ac.uk/rbf/PAPERS/unfbehavedata.pdf
[95] Rota, P., Conci, N., Sebe, N., Rehg, J. (2015) *Real-Life Violent Social Interaction Detection*
http://mhug.disi.unitn.it/wp-content/uploads/2015/papers/2015/Rota-ICIP15.pdf
[96] Demarty, C., Penet, C., Soleymani, M., G. Gravier, G., (2015) *Vsd, a public dataset for the detection of violent scenes in movies: design, annotation, analysis and evaluation*. Multimedia Tools and Applications, vol. 74, no. 17, pp. 7379–7404, 2015
https://link.springer.com/article/10.1007/s11042-014-1984-4
[97] M. Perez, M., Kot, A., Rocha, A. (2019) *Detection of real-world fights in surveillance videos.* in ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, pp. 2662–2666
https://ieeexplore.ieee.org/document/8683676
[98] Nievas, E., Suarez, O., Garcia, G., Sukthankar, R. (2011) *Hockey fight detection dataset*. Computer Analysis of Images and Patterns. Springer, pp. 332–339
http://visilab.etsii.uclm.es/personas/oscar/FightDetection/
[99] Nievas, E., Suarez, O., Garcia, G., Sukthankar, R. (2011) *Movies fight detection dataset.* Computer Analysis of Images and Patterns. Springer, 2011, pp. 332–339.
http://visilab.etsii.uclm.es/personas/oscar/FightDetection/
[100] Hassner, T., Itcher,Y., Kliper-Gross, O., (2012) *Violent flows: Real-time detection of violent crowd behavior*. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. IEEE, pp. 1–6.
https://ieeexplore.ieee.org/document/6239348
[101] Yun, K., Honorio, J., Chattopadhyay, D., Berg, T., Samaras, D., (2012) *Two-person interaction detection using body-pose features and multiple instance learning*. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. IEEE, 2012, pp. 28–35.
http://people.csail.mit.edu/jhonorio/actrec_hau3d12.pdf
[102] Sultani, W., Chen, C., Shah, M., (2018) *Real-world anomaly detection in surveillance videos*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6479–6488.
https://arxiv.org/abs/1801.04264
[103] Cheng, M., Cai, K., Li, M. (2020) *RWF-2000: An Open Large Scale Video Database for Violence Detection*
https://arxiv.org/pdf/1911.05913.pdf
[104] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z. (2015) *Rethinking the Inception Architecture for Computer Vision.* Cornell University. NY.
https://arxiv.org/abs/1512.00567
[105] Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A. (2016) *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning.* Cornell University. NY.
https://arxiv.org/abs/1602.07261
[106] Krishnan, M. (2018) *Understanding the Classification report through sklearn*
https://muthu.co/understanding-the-classification-report-in-sklearn/
Último acceso: 2020/06/10

[107] Selvaraju, R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D. (2019) *Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization*. Cornell University. NY. https://arxiv.org/abs/1610.02391

[108] Mishra, D. (2019) *Demystifying Convolutional Neural Networks using GradCam* https://towardsdatascience.com/demystifying-convolutional-neural-networks-using-gradcam-554a85dd4e48
Último acceso: 2020/06/10

[109] Rosebrock, A. (2020) *Grad-CAM: Visualize class activation maps with Keras, TensorFlow, and Deep Learning.* https://www.pyimagesearch.com/2020/03/09/grad-cam-visualize-class-activation-maps-with-keras-tensorflow-and-deep-learning/
Último acceso: 2020/06/10

[110] Chinnathambi, K. (2015) *The Ken Burns Effect using CSS Animations* https://www.kirupa.com/html5/ken_burns_effect_css.htm
Último acceso: 2020/06/10